REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 19-02-2004	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From – To)* 01-Dec-00 - 19-Feb-04

4. TITLE AND SUBTITLE Formal Methods for Information Protection Technology Task 1: Formal Grammar-Based Approach and Tool for Simulation Attacks against Computer Network Part I	**5a. CONTRACT NUMBER** ISTC Registration No: 1994p
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) O.V.Karsayev, Ph.D I.V. Kotenko, Ph.D	**5d. PROJECT NUMBER**
	5d. TASK NUMBER
	5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences 39, 14th Liniya St. Petersburg 199178 Russia	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD PSC 802 BOX 14 FPO 09499-0014	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) ISTC 00-7035

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. (approval given by local Public Affairs Office)

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences as follows: Formal Methods for Information Protection Technology
The use of open computer networks as an environment for exchange of information across the globe in distributed applications requires improved security measures on the network, in particular, to information resources used in applications. Integrity, confidentiality and availability of the network resources must be assured. To detect and suppress different types of computer unauthorized intrusions, modern network security systems (NSS) must be armed with various protection means and be able to accumulate experience in order to increase its ability to front against known types of intrusions, and to learn new types of intrusions. The project will perform three main tasks.
1. Develop a mathematical model and a tool that simulates various coordinated intrusion scenarios against computer networks;
2. Develop the mathematical foundations, architecture, and principles of implementation of autonomous-software-tool technology implementing the learning system for intrusion detection;
3. Develop the fundamentals, architecture and software for the computer security system based on multi-level encoding for information protection in mass application.
Currently, scientific efforts in network security area are undertaken mainly in the development of the network defense mechanisms. Unfortunately, substantially less attention is paid to the study of the nature of intrusions and, in particular, remote distributed intrusion attempts. No appropriate tools for intrusion/attack simulation nor research on a formal framework for intrusion specification exists.

TASK 1
The first research task in the project aims to (1) to develop a formal framework for modeling of distributed computer intrusions scenarios; (2) to develop a software tool for simulation of distributed intrusions, and (3) to explore advantages of using of such model and tool in the design and validation of the network assurance systems. Experts' analysis of distributed intrusions shows that malefactors plan attempted intrusions on macro-level as a partially ordered set of steps. Each step aims at achieving a particular sub-goal, say, to break through a "security wall", get non-authorized access to some information, services, applications, etc. The partially ordered set of the steps of intrusions on the macro level is called a scenario of attack. To realize each particular step of the intrusions scenario, the malefactor uses operations of low (micro-) level.. Thus, each such a step of the scenario is represented as a sequence of commands. Following the aforementioned conceptual representation of the intrusion attempt, the research focuses on the two-level model of attacks. It is supposed that available learning information about intrusions of different types comprises the experts' information and limited number of cases.

The importance of the Project in the framework of the ISTC mission is determined by several reasons. The Project makes it possible to involve military oriented scientists into civilian basic research. It contributes the integration of Russian scientists into international society and ministers in deciding problems of safe and secured utilization of the network, in particular, Internet-based information resources.

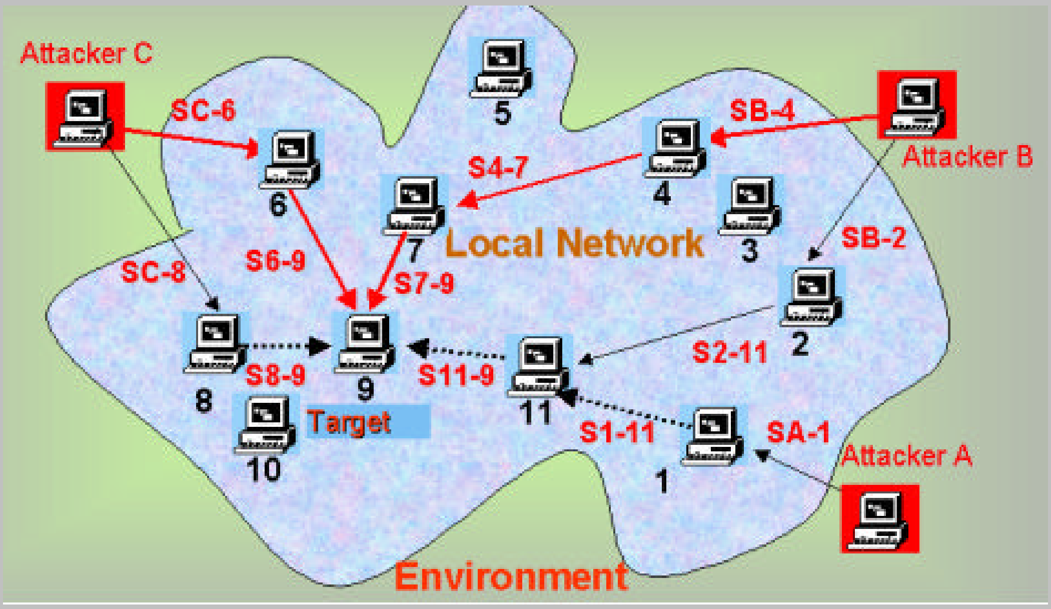
15. SUBJECT TERMS
EOARD, Mathematical & Computer Sciences, Computer Systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** /Signed/PAUL LOSIEWICZ, Ph. D.
a. REPORT UNCLAS	**b. ABSTRACT** UNCLAS	**c. THIS PAGE** UNCLAS	UL		**19b. TELEPHONE NUMBER** *(Include area code)* +44 20 7514 4474

ST. PETERSBURG INSTITUTE FOR INFORMATICS AND AUTOMATION

EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT (EOARD)

Project 1994P
Formal Methods for Information Protection Technology



Final Report
Task 1: Formal Grammar-Based Approach and Tool for Simulation Attacks against Computer Network
Part I

Project Manager
Research Fellow of SPIIRAS
Ph.D. O.V.Karsayev
Manager of Task 1
Leading Scientist of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg
February, 2003

ST. PETERSBURG INSTITUTE
FOR INFORMATICS AND AUTOMATION

(SPIIRAS)

EUROPEAN OFFICE OF AEROSPACE
RESEARCH AND DEVELOPMENT

(EOARD)

Formal Grammar-Based Approach and Tool for Simulation of Attacks against Computer Networks

Final Report

on Task 1 of the Project # 1994P

Part I

Project Manager
Research Fellow of SPIIRAS
Ph.D. O.V.Karsayev
Manager of Task 1
Leading Scientist of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg

February 2003

Contents

Preface

This volume is the Final Report on the *Task 1 of the Project #1994P "Formal Methods for Information Protection Technology"* that is being performed according to the agreement between European Office of Aerospace Research and Development (EOARD), The International Science and Technology Center (ISTC) and St. Petersburg Institute for Informatics and Automation (SPIIRAS).

Task 1 of the Project #1994P is entitled *"Formal Grammar-Based Approach and Tool for Simulation of Attacks on Computer Network"*. This report describes the results of the fifth research phase scheduled by the Work Plan and also summarizes results of the Project research on the whole.

Formal model of distributed attacks is the subject of the research presented in this report. The goal of the Task 1 of the Project is *development of the formal model and software for simulation of broad spectrum of network attacks, and also an investigation of their possibilities and usefulness in analyzing of computer network assurance*.

According to the Work Program, at this phase of research the following task is scheduled:

A-4. Development of the software prototype of the Attack Simulator implementing theoretical results of research and its evaluation.

The results are presented in two chapters associated with the aforementioned tasks.

Chapter 1 "*Overview of the theoretical results presented in previous reports: formal grammar-based approach for modeling and simulation of computer network attacks*" summarizes the suggested approach for modeling attacks against computer network, the developed technology and software tool for design and implementation of knowledge-based multi-agent systems, and the object-oriented project of the Attack Simulator. More detailed description of these results was given in Interim Reports submitted to EOARD according to the Work Program ([IntRep#1], [IntRep#2], [IntRep#3]).

In Chapter 2 "*Software prototype of the Attack Simulator implementing theoretical results of the research and their evaluation*", the results of research on the task A-4 are presented. The Chapter describes the architecture and main components of the Attack Simulator prototype, as well as its functional capabilities and specific features of implementation. It also outlines the simulation-based exploration of the developed Attack Simulator prototype and its benefits in use for evaluation of the computer network system assurance.

All theoretical results and conclusions of the research are explored and validated via simulation on the basis of the software developed by authors. The developed software can be demonstrated in AFRL/IT as well as software code can be submitted to the Partner on demand.

All tasks presupposed by the Work Program are solved completely.

The papers describing results of the Project have been accepted for presentation and publication in Proceedings in several Russian and International Conferences ([Gorodetski *et al*-01a], [Gorodetski *et al*-01b], [Gorodetski *et al*-01c], [Gorodetski *et al*-01d], [Gorodetski *et al*-02a], [Gorodetski *et al*-02b], [Gorodetski *et al*-02c], [Gorodetski *et al*-02d], [Gorodetski *et al*-02e], [Gorodetski *et al*-02f], [Kotenko *et al*-02a], [Kotenko *et al*-02b], [Kotenko-02a], [Kotenko-02b], [Alexeev *et al*-02], [Stepashkin *et al*-02], [Nesterov *et al*-02], [Kotenko *et al*-03], [Kotenko-03]), including *Fifth International Symposium "Recent Advances in Intrusion Detection (RAID 2002)"* (Zurich, Switzerland. October 2002) [Gorodetski *et al*-02c], *Fourth International Workshop "Agent-Based Simulation 4 (ABS 4)"* (Montpellier, France. April 28-30. 2003) [Kotenko *et al*-03] and *Third International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)* (Prague, The Czech Republic. June 16 – 18, 2003) [Kotenko-03].

Two papers have been published and one has been accepted for publication in the *Lecture Notes in Computer Science and Artificial Intelligence* series ([Gorodetski *et al*-02a], [Gorodetski *et al*-02c], [Kotenko-03]). One paper has been prepared for publication in *IEEE Security and Privacy* journal.

Project manager

Leading Scientist of the St. Petersburg Institute for Informatics
and Automation of the Russian Academy of Sciences

Ph.D. Prof. Igor Kotenko

Chapter 1. Overview of the Theoretical Results Presented in Interim Reports: Formal Grammar-based Approach for Modeling and Simulation of Attacks against Computer Networks

Abstract. This chapter describes theoretical results that mostly already have been presented in Interim Reports submitted to EOARD according to the schedule supposed by Work Program and gives a review of recent related works. It presents a brief survey of computer network attacks, including definitions of main concepts, a review of existing computer and network attack taxonomies, classification and analysis of standard remote attacks. It introduces the scenario-based specifications of computer network attacks. Specifications are based on developed conceptual model of attacks and proposed generalized formal means for attack scenario specifications. An important task concerning synthesis (recovery) of grammars specifying models of attacks is analyzed and the corresponding grammar recovery algorithms are described. The developed mathematical methods and techniques used for formal modeling of attacks are reviewed. In this research attack model is considered as a complex process of contest of adversary entities those are malefactor or team of malefactors, on the one hand, and network security system implementing a security policy, on the other one. The Chapter also presents conceptual justification of the chosen approach, specification of the basic components composing attack model and their interaction in simulation procedure and describes examples of the network attacks specifications. The behavior of malefactor implementing an attack is specified in terms of state machines simulating attack development. An important issue is formal model of the attacked computer network and its response to attacks. This model is developed and described in the Chapter. Implementation and deployment of the agent-based attack simulator was carried out by use of "Multi-agent System Development Kit", MAS DK, which is outlined. The final design result that concerns the object-oriented project of the software prototype of the Attack Simulator is overviewed in the end of the Chapter.

1.1. Introduction

Efficiency of computer network security systems, including intrusion detection systems, vulnerability assessment kits, honeypots, etc., depends in a high degree on the quality and completeness of the knowledge about strategies and implementation of computer network attacks taken into account in the security policy.

However, modern computer network security systems use mostly "ad hoc" built security policies aimed at defense against *known types* of attacks and other threats. It is undoubtedly that remarkable increase of security systems efficiency could be achieved in case of using knowledge resulting from generalization and formalization of the accumulated experience regarding computer system vulnerabilities and attack cases data ([Axelsson 00], [Allen *et al*-00], [McHugh-01]).

Till now a lot of such data is accumulated. There are a number of publications in which the attack cases are systematized in the form of taxonomies ([Aslam-95], [Howard *et al*-98], [Krsul-98], [Ranum-97], etc.). Nevertheless, till now there are no serious attempts to generalize the accumulated data in order to develop a *formal model* of computer network attacks. Such a model could be a powerful source of knowledge needed for security systems development and implementation. It could help in *deeper study of the essence and peculiarities of attacks* (intentions of malefactors, attack objects, structure of attacks, strategies of attack realization, etc.).

The current competition between security systems and malefactors, in which the latter are permanently inventing new attacks, is such that malefactors have a remarkable advantage. This advantage is becoming more noticeable if malefactors implement distributed attacks intending as a target not only a particular host but also the whole computer network. An example of such an attack is given in [Mukherjee-94]. It exemplifies a distributed attack that comprises 11 phases performed during several days. At present, it is not possible to detect such attacks automatically and particularly on-line. However, in the modern days distributed attacks are becoming the practice of malefactors.

This is a cogent argument for the necessity of deep study and research of the essence and peculiarities of distributed attacks. The study cannot only be restricted by generalization of the experience; it has also to be based on using of formal models and simulation of attacks. The intentions and objects of attacks, strategies and ways of realization must be the prime subjects of such a study. *These formal models could be very valuable in the design of security systems capable to operate with high-level notions like "identification of an attack scenario", "forecasting of the attack development"*, etc. Such capabilities could make feasible to break on-line an attack development before the irreversible consequences. "We also need to know what the exploited vulnerability is, how the attack was performed, what are its consequences, and how to react (automatically or not) in order to stop it [Michel *et al*-01]."

The model of distributed attacks could be very useful in learning to cope with both known and unknown attack detection. Therefore, *artificially generated sample of attack could be used as training and testing data for security system learning, especially intrusion detection systems learning.*

Finally, *a formal model of attacks and attack simulation tool if used together with* vulnerability assessment systems *could play an important role in the validation of security policies*. Such simulator could be used as a test bed for security systems thus providing decrease of the cost and time of a security policy validation.

A formal model of computer network attacks is the subject of the research presented in the Project. The *goals of the Project* are the following:

(1) development of a powerful formal framework for specification of a broad spectrum of attacks against computer network;

(2) elaboration of formal specifications of a representative spectrum of such attacks;

(3) implementation of a software tool prototype making it possible to simulate attacks and respective responses of the attacked computer network objects;

(4) exploration of practical utility of Attack Simulator prototype.

According to the Work Program the Project research was carrying out in several phases and results of each stage were submitted in three Interim Reports ([IntRep#1], [IntRep#2], [IntRep#3]) and in this, Final report.

The *Final Report* is concluding one. It gives a brief summary of the results presented in the previous reports in order to connect them with the results presented in this one, and to make available the complete understanding of the results on the whole.

Besides the Final Report presents results, which are a natural continuation and also complement to the previous results, that is software prototype of the Attack Simulator implementing theoretical results of research and its evaluation.

1.2. Specification of the representative set of distributed attacks against computer networks

1.2.1. Analysis and classification of attacks on computer networks

A review of computer network attacks was given in *the Interim Report #1* [IntRep#1]. This review includes the following main elements:

- Definitions of the main concepts of remote attacks on computer networks;
- Review of existing computer and network attack taxonomies;
- Classification of standard remote attacks;
- Analysis of remote attacks on computer networks.

The *purpose of computer networks attacks* undertaken by the malefactors (intruders) consists in obtaining access to the necessary information and network resources, violation of their integrity and availability. A *basic feature of attacks* realized by the malefactors in open networks is a factor of distance between a host selected as a victim and a malefactor. So a *remote attack* is an unauthorized information effect on objects of the computer networks realized on data links.

Invariance of mechanisms of an attack implementation with regard to features of the concrete system (topology, infrastructure, type of a network operating system, protocols of an interaction) allows using a concept of a *standard remote attack* as a remote effect irrespective to the type of a computer networks. The majority of computer networks, including the Internet, were formed as unprotected systems, which have not been intended for storage and processing of confidential information.

On the basis of the analysis of many scientific and technical materials concerning the network security the following *taxonomies of network attacks* were analyzed ([Radatz *et al*-96], [Krsul-98], [Landwehr *et al*-94], [Amoroso-94], [Howard-97], [Howard *et al*-98]):

(1) Lists of attack terms ([Cohen-95], [Icove *et al*-95], [Cohen-97], [Howard-97], [Howard *et al*-98]),
(2) Lists of attack categories ([Cheswick *et al*-94], [Ranum-97]),
(3) Attack results categories ([Cohen-95], [Russell *et al*-91]),
(4) Empirical lists of attack types ([Lackey-74], [Neumann *et al*-89], [Amoroso-94], [Lindqvist *et al*-97]),
(5) Vulnerabilities matrices ([Amoroso-94], [Landwehr *et al*-94]),
(6) Action-based taxonomies [Stallings-95],
(7) Security flaws or vulnerabilities taxonomies ([Beizer-90], [Saltzer *et al*-75], [Hogan-88], [Aslam-95], [Dodson-96], [Krsul-98], [Power-96]),
(8) Taxonomies of intrusions based on the signatures [Kumar-95],
(9) Incident taxonomies ([Howard-97], [Howard *et al*-98]).

These taxonomies are multifold but can not ensure the project objective realization. We think that the incident taxonomies are the most prominent for our goals, but they require a more elaboration with emphasis on remote actions.

On the basis of generalization of the works on computer network security ([Radatz *et al*-96], [Krsul-98], [Landwehr *et al*-94], [Amoroso-94], [Howard, 97], [Howard *et al*-98], [Medvedovsky *et al*-99], [Cole-02], et al) *classification of standard remote attacks* is developed. According to this classification, *the remote attacks are structured by seven basic tags*:

- Character of an effect (passive, active);
- Purpose of an effect (violation of confidentiality, integrity and service availability);
- Condition of beginning of the effect realization (according to an inquiry from the attacked object, on fulfillment of the expected event on the attacked object, unconditional attack);
- Availability of feedback with the attacked object (with feedback, without feedback – one-direction attack);
- Layout of the subject of attack concerning the attacked object (intra-segmental, inter-segmental);
- Layer of standard ISO/OSI model, on which the effect is carried out (physical, link, network, transport, session, presentation, application);
- Object on which an effect is directed to (on network services, on an infrastructure of the network);
- Attack complexity level (simple and composed).

Let us consider main classes of standard remote attacks according to aforementioned basic tags.

1. *Character of an effect* on the distributed computing system or network*:*

(1) *Passive effect*;

(2) *Active effect*.

The *passive effect* is one, which does not render immediate influence on system operation, but can break its security policy. The *active effect* is one, rendering immediate influence on the system operation (reconfiguration of a system or network, violation of a service capability, etc.) and breaking its security policy. The example of attacks of the first type is listening of data links and interception of information entered from the keyboard. The example of the second type of attacks is an attack "third in the middle", when the malefactor can substitute data of the message exchange between two users of the network or between the user and the network service requested by him.

2. *Purpose of an effect:*

(1) Violation of information confidentiality;

(2) Violation of information integrity;

(3) Violation of service capability (availability) of the system.

This classification attribute is a direct projection of three main types of threats - disclosure, integrity and denial of service.

3. *Condition of beginning of the effect realization*:

(1) *Attack according to an inquiry from the attacked object*. In this case an intruder (attacker) expects transmission from the potential attack object of defined inquiry, which will be the condition of beginning of the effect realization;

(2) *Attack on fulfillment of the expected event on the attacked object*. In this case an intruder realizes continuous tracing of the state of an operating system of the remote host, and starts the effect at origin of a specific event in this system;

(3) *Unconditional attack*. In this case the beginning of attack realization is unconditional with regard to the attack purpose that is the attack is carried out immediately and regardless to state of attacked object.

4. *Availability of feedback with the attacked object*:

- Feedback from attacked object is provided;
- No feedback (one-direction attack).

Under remote attack, realized at presence of *feedback with the attacked object*, the intruder must receive answers to some inquiries transferred to the attacked object. Therefore between the intruder and the attacked object a feedback exists, which allows intruder to react adequately to all changes happening on the attacked object. The remote attacks *without feedback* do not require reacting to any changes happening on the attacked object.

5. *Layout of the subject of attack concerning the attacked object:*

(1) Intra-segmental;

(2) Inter-segmental.

Segment of the network is a physical association of hosts. For example, a segment of the network can consist of a set of hosts connected with the server by means of the "common bus" scheme. The *intra-segmental attack* is an attack, when the subject and object of attack are situated in one segment. The *inter-segmental attack* is an attack, when the subject and object of attack are situated in different segments.

6. *Layer of standard ISO/OSI model* on which an effect is taken:

(1) Physical layer;

(2) Link layer;

(3) Network layer;

(4) Transport layer;

(5) Session layer;

(6) Presentation layer; and

(7) Application layer.

7. According to *the object, on which an effect is taken,* two classes of attacks can be discerned:

(1) *Class A* – attacks on network services (effects on an application layer);

(2) *Class B* – attacks on an infrastructure of the network (effects on layers below the application layer).

The *attacks of class A* are directed on lowering of efficiency of the computing systems operation by means of effects on application processes. These attacks have the following purposes: degradation of the workstations and servers performance or violation of their service capability; obtaining of an unauthorized access to information (violation of confidentiality, integrity and availability of information processed by application processes); imposing of false information at an interaction of application processes through data links.

The *attacks of class B* are directed on lowering of efficiency of the network operation through effecting on the network infrastructural characteristics. The purposes of these attacks are the followings: deterioration of dynamic characteristics of a telecommunication subsystem (reduction of data links capacity, degradation of network devices or violation of their service capability); change of the network structural characteristics (change or violation of logical connectivity between objects, implantation of false objects).

8. *Attack complexity level*:

(1) Simple;

(2) Composed (complex).

Simple attacks consist of one or several actions. *Complex attacks* include a set of the simple attacks.

The enumerated characteristics of attacks can be considered as a basis for construction of complex attack classifications necessary for the concrete applications.

We discern eight <u>*typical classes of the remote attacks*</u>:

(1) Analysis of the network traffic;
(2) Network scanning (probing);
(3) Substitution of the trusted object of the network and transmission of the messages from its name with appropriation of its access rights;
(4) Implantation of the false object in a network;
(5) Denial of service;
(6) Unauthorized access from a remote machine by guessing password;
(7) Unauthorized access to local superuser (root) privileges;
(8) Remote initiation of applications.

The eight most frequently undertaken *typical classes of the network attacks and their examples* are represented in Tab.1.2.1.

Tab.1.2.1. Classes of network attacks

Attack type	Attack Examples
1. Analysis of the network traffic – sniffing or listening of a data link by means of sniffers	▪ to study a logic of the network operation (to get unambiguous correspondence of events happening in the system and commands transferred by hosts) ▪ to intercept a stream of data (for extraction passwords for access to the remote hosts on the FTP and TELNET protocols) *Means*: Esniff.c, Gobler, ethdump, LanPatrol, LanWatch, Netmon, Netwatch, ethload, Linsniffer, BUTTSniffer, Session Wall-3, LANAlyzer, PacketBoy, Lan Trace, Shomiti Surveyor, Sniffer Ballista/NT, tcpdump, web_snif.c, readsmb.c, icq-spof.c, C2MyAzz, IP-Watcher, etc. ▪ to study symbol sequences entering into the host from the keypad *Means*: Keytrap, Playback, Keycopy, Getit, etc.
2. Network scanning (probing) – a transmission of inquiries to the network services of hosts and analysis of the answers from them	*Means*: ipsweep, mscan, portsweep, satan, ping, fping, Pinger, WS_PingProPack, icmpquery, icmpush, strobe, udp_scan, netcut, PortPro, nmap, ident, queso, cheops, tkined, etc.
(1) ”Visible” scanning	▪ TCP ports scanning (TCP SYN scanning, TCP FIN scanning, scanning based on IP fragmentation, TAP IDENT scanning) ▪ UDP ports scanning ▪ Scanning by DNS ▪ Scanning by ping sweep
(2) ”Invisible” anonymous scanning	▪ Half scan ▪ FTP bounce ▪ Dumb host scan ▪ Proxy scanning ▪ Scanning by ministorm of inquiries
3. Substitution of the trusted object of the network and transmission on links of the	

messages from its name with appropriation of its access rights	
(1) Substitution of the trusted object of a network at an establishment of virtual connection	▪ TCP-IP Spoofing ▪ Web spoofing
(2) Substitution of the trusted object of a network without an establishment of virtual connection	▪ DNS spoofing
4. Implantation of the false object of the network	
(1) by obtrusion of the false path using the disadvantages of routing algorithms of the protocols	Attacks using RIP, OSPF, LSP, ICMP, SNMP protocols
(2) by usage of disadvantages of the remote search algorithms	Attacks using SAP, ARP, DNS, WINS protocols
5. Denial of service (DoS)	
(1) *DoS caused by usage of a portion of the network resources*	▪ directed storm of echoes - inquiries on the ICMP protocol (Ping flooding) ▪ ministorm of inquiries on installation of TCP-connections (SYN flooding) ▪ storm of inquiries to FTP server
(2) *DoS caused by exhaustion of the network resources at processing of packages transmitted by the malefactor*	▪ storm of broadcasting ICMP- echoes-inquiries (Smurf) ▪ directed storm (SYN flooding) ▪ storm of the messages to a mail server (Spam)
(3) *DoS caused by violation of logical connectivity between the network objects* by transmission of control messages changing the route data or the identification and authentication information on behalf of network devices	▪ ICMP Redirect Host, ▪ DNS-flooding
(4) *DoS caused by transmission of packages with uncommon attributes or having length exceeding a valid maximum size*	▪ Land, TearDrop, Bonk, Nuke, UDP bomb ▪ Ping Death, attack on a ftpd demon of UNIX host
6. Unauthorized access from a remote machine by guessing password	▪ “brute force” method; ▪ simple guessing password; ▪ “crypt and compare” method ▪ social engineering
7. Unauthorized access to local super user (root) privileges	various “buffer overflow” attacks
8. Remote initiation of applications	
(1) distribution of files containing the unauthorized executed code	I-Worm.LoveLetter
(2) remote initiation of an application by buffer overflow of the application server	Morris virus
(3) remote initiation of an application by usage of possibilities of the remote system control provided by hidden software and hardware beetles or used regular means	Back Orifice, Net Bus, Landesk Management Suite, Managewise, BackOffice

The analysis of the network traffic consists in an interception of network packets and their analysis. This attack class allows to research the parameters of a network (protocols, topology, types of operating systems, physical and logic addresses of objects) and to get access to the confidential information (for example, to the users' names and passwords). The widespread type of the data interception means is a sniffer, i.e. a network analyzer or means of dataflow inspection. Examples of this type programs are the following: for SunOS - Esniff.c (it captures only first 300 bytes telnet, ftp and rlogin sessions, that it is quite enough for obtaining the identifier and password), Etherfind and Snoop; for the MS DOS - Gobler, ethdump, LanPatrol, LanWatch, Netmon, Netwatch, ethload; for Linux - Linsniffer; for Windows - BUTTSniffer V 0.9.3, Session Wall-3, LANAlyzer, PacketBoy vl.2 for Win95/NT, Lan Trace, Shomiti Surveyor, Sniffer Ballista/NT, etc. For many operating systems, the Unix utility tcpdump can be used.

The network scanning consists in a transmission of inquiries to the network services of hosts and analysis of the answers from them. The purpose of the attacks of this class is detection of the used protocols, accessible ports of network services, determination of active network services, selection of users' identifiers and passwords. The basic ways of scanning are: TCP ports scanning (for example, TCP SYN-scanning, TCP FIN-scanning, scanning based on IP fragmentation, TAP IDENT scanning, "Christmas tree" scanning, zero scanning), UDP ports scanning, scanning by DNS, ping sweep scanning, "proxy"-scanning, FTP bounce scanning, etc. The most widely used means of network scanning are nmap, strobe, udp_scan, netcat, PortPro, Portscan, Ipsweep, Mscan, Portsweep, Satan, etc.

The substitution of the trusted object of the network and transmission of the messages on its behalf and appropriation of its access rights is effectively realized in systems, where unstable algorithms of the identification and authentication are used.

Two varieties of the attacks of this class can be discerned:

(1) attacks using installed virtual connection;

(2) attacks without installation of a virtual connection.

The first type of attack consists in appropriation of the rights of the trusted subject of interaction that allows intruder to perform a session with the network object on behalf of the trusted subject (for example, attack of the rsh-service of a UNIX-host). An attack without installation of virtual connection can take place in networks realizing an identification of the transmitted messages only using a network address of the sender. The essence of this attack consists in transmission of service messages concerning the change of the route data on behalf of the network control devices. An example of such attack is imposing of the false router using of the ICMP message "Redirect Host".

The main purpose of *implantation of the false object in the network* is a variation of the route data on the attacked object so that the new path passes through the false object. This attack is carried out in two ways:

(1) by obtrusion of the false path using disadvantages of routing algorithms. As a result, the attack object traffic can get, for example, to the host of the malefactor, where it is possible "to open" the attacked host by means of some tools. This way consists in unauthorized use of the routing protocols (RIP, OSPF, LSP) and the network management protocols (ICMP, SNMP) for modification of the route data;

(2) by use of disadvantages of the remote search algorithms. If the network objects have no address information about each other, various protocols of the remote search are used (for example, SAP in Novell NetWare networks; ARP, DNS, WINS in TCP/IP networks). The remote search protocols consist in transmission to the network of address retrieval inquiries and obtaining the answers with the required information. Thus, the intruder can intercept the address retrieval inquiry and transmit the false answer, whose use will change the route data. Further all traffic associated with the object-victim will pass through the false network object. This attack allows to effect on the intercepted information as follows: to carry out selection and saving of data stream; to update transmitted data or transmitted code; to substitute transmitted data.

"Denial of service" attack consists in transmission by intruder on behalf of the legal objects of many inquiries addressed to network services, or transmission of packages with unusual attributes, or having length exceeding a valid maximum size. Some varieties of this class of attacks can be indicated: (1) the hidden denial of service caused by the use of a part of computer network resources

while processing the packages transmitted by the malefactor. That can violate the requirements for the inquiry processing time. Examples of these attacks are as follows:

(1)directed storm of echoes - inquiries on the ICMP protocol (Ping flooding), mini-storm of inquiries on installation of TCP connections (SYN-flooding), storm of inquiries to FTP server;

(2) the evident denial of service caused by exhaustion of the network resources at processing of packages transmitted by the malefactor. Examples of these attacks are a storm of broadcasting ICMP-echoes-inquiries (Smurf), directed storm (SYN-flooding), storm of the messages to the mail server (Spam);

(3) the evident denial of service caused by violation of logical connectivity between the network objects by transmission of control messages changing the route data or the identification and authentication information on behalf of network devices (for example, ICMP Redirect Host, DNS-flooding);

(4) the evident denial of service caused by transmission of packages with unconventional attributes (Land, TearDrop, Bonk, Nuke, UDP-bomb) or having length exceeding the valid maximum size (Ping Death, attack of a ftpd demon of UNIX-host). This attack can cause a failure of network devices participating in the inquiry processing in presence of faults in programs realizing the network exchange protocols.

Unauthorized access from a remote machine by guessing password can be realized in three ways: (1) “brute force”; (2) simple guessing; (3) “crypt and compare”. Besides, social engineering can be used. When “brute force” method is used, an attacker, first of all, can test default passwords installations (for example, in Unix-systems - root or bin, in VMS - system, in Windows NT - administrator, in Netware - supervisor), or guest passwords (guest, demo, visitor). When simple guessing password is realized, the special programs automating this process are used. Some of these programs use a list of widespread passwords with known or installed "on default" login name, others apply network utilities of the user’s login determination (Finger for Unix, Finger32 and WSFinger for the Windows, FFEU for OS/2 etc.), and try as passwords various permutations of symbols in these names. Malefactors, as a rule, use password guessing means based on “crypt and compare” method. It consists in an encryption of various words via the algorithm used for encryption of the passwords, and matching two ciphered strings. If they are equal the necessary password is determined.

Unauthorized access to local super-user (root) privileges is carried out, as a rule, by start-up of the application, which causes buffer overflow under the preset initial conditions. In some cases of the system registers adjustment the processor can be switched after interruption caused by buffer overflow to fulfillment of a code contained out of the buffer space and possessing higher rights. The attack of this type was used in a well-known Morris virus.

“Remote initiation of applications” attacks consist in an implantation and initiation of various beetles on the attacked host (for example, trojan programs Back Orifice, Net Bus), viruses (for example, “VBS.LoveLetter”), or in use of a standard network control and administration means (Landesk Management Suite, Managewise, Back Office, etc.). The main purpose is violation of information confidentiality, integrity, availability and complete administrative control of the host operation. Schematically the main stages of these programs operation are as follows: installing in memory; waiting for inquiry from the remote host, on which the head server program is initiated; message exchange with the head server program; transmitting the intercepted information to the head server-program or granting it control of the attacked host.

The analysis of mentioned classes of remote attacks is elaborated in detail in *the Interim Report #1* [IntRep#1]. The represented results allowed developing the conceptual descriptions of representative set of network attacks and their formal models.

1.2.2. Scenario-based specification of the representative set of distributed attacks of different classes

The scenario-based specifications of computer network attacks were given in *the Interim Report #1* [IntRep#1].

It was shown that computer network attacks are of great concern to the class of complex systems possessing such features as large scale, multi-connectivity of elements, diversity of their connections, variability of structure, multiplicity of executed functions and structural redundancy.

An *attack model* is understood as a formal object having a likeness in basic properties with regard to real-life attacks, serving for investigations of their properties by means of using known and obtaining new information about attacks. A *formal model* of attacks is a collection of mathematical dependencies specifying attacks and allowing study of them formally and via simulation.

The research focused on the <u>*conceptual model of attacks*</u> which includes two levels: (1) macro-level and (2) micro-level. It is supposed that available information about attacks to study them comprises experts' information and limited number of cases.

Analysis of remote attacks proved that a malefactor plans each attack on *macro-level* as a partially ordered set of steps. The partially ordered set of attack steps on the macro level is called a *scenario of attack*. In any case, realization of a scenario is represented by a sequence of various lengths. Each step aims at achieving a particular sub-goal, say, to break through a firewall, to get a non-authorized access to some information, services, applications, to execute an operation with the object of interest, to remove evidences of the attack steps, etc. In realization of some steps of such a scenario may not be successful, while other ones may be successful. In principle, the total number of such steps of different purposes is not too large, and they can be realized by malefactors in diverse orders, in a repeatable mode, from different source hosts, etc. Availability of even a unique case of an attack scenario allows an expert to identify and anticipate the malefactor's intentions, peculiarities of the attack implementation and to anticipate a variety of possible scenarios aimed to the same target

To realize each particular step of the attack scenario, the malefactor uses operations of low level. Therefore, each such a step of a scenario may be represented as a sequence of low level commands, system calls, etc., that specify an attack on *micro-level*. Although each step can be formed by the same operations, specific character of each step can be expressed by particular probabilistic characteristics of sequences of commands corresponding to the various steps of scenario and by particular values of attributes (names of files, directories, services, etc.).

The developed <u>*means of the attack specification*</u> can be represented as described below.

On macro-level, each sequence of attack steps may be considered as a “word” belonging to a formal language that, in turn, can be specified by a formal grammar ([Aho *et al*-72], [Fu-74], [Gorodetski-86], [Lammel *et al*-00]). A description of the common attack scenario by stochastic grammar which is the following[1]:

$$GA=\langle V_N, V_T, S, P\rangle,$$

where V_N is a set of the non-terminal symbols, which are put into the correspondence with the upper levels of attack steps; V_T is a set of the terminal symbols, which denote the attack steps of the lower level; S is an initial symbol ("axiom") of attack scenario; P is a set of the productions assigned probabilities of their use. P represents syntax of the language to be generated by grammar GA, at that each "word" of this language represents the sequence of malefactor's activity on macro-level.

It is supposed that each terminal symbol of the "word" of the language generated by grammar GA, in turn, can be considered as an "initial symbol" (axiom) of a grammar specifying the respective step in more details. Formally, this more detailed elaboration of the attack specification corresponds to the well known (within formal grammar theory) operation called "*Substitution of grammar*" ([Glushkov *et al*-78]).

The developed model of attack scenario is very flexible. It is able to specify such peculiarities of attacks as variety of orders of steps of the same attack in different its instances, repeatability of steps, possibility to initiate different steps of an attack from different hosts (if attack is performed in distributed way), etc. It also takes into account that attack can be directed to different objects of the victim computer network.

This grammar may be regenerated by formal methods inductively on the basis of cases, and later it can serve as a formal model of such kind of attacks on the macro level. This grammar can play a dual role, namely, it can be used as model of cases generation, and it can serve as a formal model used for attack detection on the basis of syntactical analysis of the "words" representing scenario of the malefactor's activity. There also exist other options of formal approaches to modeling of attacks, e.g., Markov’s Chains model.

[1] Later in subsection 1.3.3 a more precise definition of the grammar used for attack specification is done.

In terms of micro-level, each step of the macro level scenario consists of a sequence of events (for example, system calls). Modeling and simulation of an attack on the micro level can also be realized by means of the expert analysis of the intruder's intention at each step of the attack. In some respects this task is similar to the one considered on the macro level. But as a rule, intruder's actions on the micro level may slightly vary from the normal actions and be more noticeable on the macro level. This is a significant argument in favor of the necessity of the macro level attack modeling and simulation. Mathematically the formal model on micro level can also be specified in terms of formal grammars or in terms of Markov's chains. Let us remind that operations of micro level are used for implementation of each step of the macro-level attack scenario.

The *scenario-based models* are practically specified for the following classes of network attacks: (1) analysis of the network traffic, (2) network scanning (probing), (3) substitution of the trusted object of the network and transmission of the messages from "on its behalf" with appropriation of its access rights, (4) implantation of the false object in a network, (5) denial of service, (6) unauthorized access from a remote machine by guessing password, (7) unauthorized access to local super user (root) privileges, and (8) remote initiation of applications.

Each scenario is described in terms of a set of admissible sequences of steps specifying attack class on macro- and micro levels.

The models of the "*analysis of the network traffic*" of attacks include a sequence of the following stages: indication of the place in the network from which it is favorably to listen the network; determination of analyzed levels of network protocols and the protocols themselves; determination of the running network equipment in the network and mechanisms of its operation; determination of the software for analysis and OS under control of which this analysis will be realized; adjustment of the software and the development of rules (patterns) on which basis information is filtered; analysis and choice of host masking means, when an intruder analyzes the traffic; intrusion in the network and starting up of all software (both analyzing the network traffic, and masking the intruder); reception and analysis (filtering) of the traffic passing through the intruder's network; disconnecting from a network; analysis, decoding, and classification of the information received by intruder.

The most important stages, which can be presented in *network scanning*, are the followings: selection of an "agent" computer and connection to it; finding computes existing in the target network; recognition of the target network structure; recognition of the services running on the target computer; getting additional information about the target network.

The common stages of the attack "*substitution of the trusted object of the network*" *are*: preparatory stage concerning analysis of the attacked objects and substitution of information on the server; listening of the network; sending of a query (a storm of queries); sending of a reply, mathematical prediction of the next message number and its sending to the attacked host, rerouting the query on the intruder's host by the server; execution of commands on the attacked host; reception and analysis of the intercepted information; influencing on intercepted information; transfer of intercepted information (probably changed or substituted); distribution of attack on other objects.

The models of the *"implantation of the false object of the network"* attacks include the following *stages*: studying the attacked host network; listening of the network; sending a false message (or a storm of messages); reception and analysis of intercepted information by the intruder or the "deceived" server; influencing on intercepted information; transferring of intercepted information (probably changed or substituted).

The models of the "*denial of service*" attacks contain a sequence of the following generalized stages: a reconnaissance of the network; an installation of master-agents and daemon-agents on the intermediary (auxiliary) hosts; sending messages from daemon-agents to master-agents (for example, about the status); sending information about the status of daemon-agents from master-agents to a malefactor; a sending of commands from the malefactor's host to master-agents; sending of commands from master-agents to daemon-agents; sending of a specially crafted packet from the malefactor's host (or from the host used by the malefactor) to the intermediary host (or a set of intermediary hosts); the intermediary host (or a set of intermediary hosts) receives the packet and responds by sending a packet to the target host; the target host receives the packet and responds back to the intermediary host; sending of a specially crafted packet (a sequence of the packets, fragments of the packet) from the malefactor's host (from the host used by the malefactor or by daemon-agents) to the target host.

The main stages of the *"unauthorized access from a remote machine by guessing password"* attacks are: getting information about the target system and its authentication subsystem; getting information about users of the target system; interception of ciphered (or hashed) passwords; getting database with ciphered (hashed) passwords; single entering of the password in online mode; multiple entering of passwords in online mode; retrieval of the passwords in offline mode; interception of passwords in plain text format (may be used for some other services).

The following stages are common for *"unauthorized access to local super user (root) privileges"* attacks: analysis of the attack targets; preparation of the code; implantation of the code; implantation of parameters ("parameterization" of the code); transfer of control to the code.

The *"remote initiation of an application"* attacks are characterized by the following stages: reconnaissance of the target computer system; implementation of a malicious code or program text into the target system; unauthorized access to the system resources; initiation and usage of auxiliary software, which is legally installed in the target system; initiation of a malicious program; activation of some special functions, available in the implemented malicious program; sending of information from the implemented program to the intruder; cleaning logfiles and deleting other attack evidences; self-reproduction of a malicious program.

For representation of these models (see *Interim Report #1*, [IntRep#1]) the scenario-based specification including verbal and formalized representation of attack steps was used.

1.2.3. Techniques for case-based regenerating of the formal grammar specifying models of the attacks

Analysis of the task of sample-based synthesis (recovery) of grammars specifying models of attacks was presented in *the Interim Report #2* [IntRep#2]. The respective grammar recovery algorithms were described. In order to demonstrate the expressive and performance capabilities of formal grammar-based model of attacks against computer networks, several cases of computer network attacks were conceptually analyzed and specified. Examples of use of grammar recovery algorithms for specification of computer network attacks were also considered.

Analysis of sample-based of synthesis (recovery) of grammars specifying models of attacks showed that the scenario of a computer network attack can be represented in terms of formal grammar. This grammar can be used both as a model generating the instances of attacks and as a model for recognition of attacks based on syntactic analysis of sequences of malefactor's steps. For practical implementation of scenario-based attack simulation systems, it is possible to construct such grammars on the basis of cases of attacks.

Formally, synthesis of a formal grammar consists in sample-bases recovery of an unknown grammar productions in which the role of "sample" plays a finite set of words *S* of the language *L(G)* to be recovered as well as possibly a finite set of words from the supplement to the language *L(G)* ([Fu-74], [Gorodetski-86], [Lammel *et al*-00], etc.).

If the task of grammar recovery solved ambiguously it would be advisable to use some quantitative measure (metric) of the grammar recovery quality, the values of which could help compare different admissible solutions. Typically, this measure characterizes the complexity of resulting grammar, and its specific form should take into account the peculiarities of the specific applied task at hand and the possible "losses" resulting from the inaccuracy of the recovered grammar.

Three different approaches applicable to the synthesis of grammar that generates scenarios for computer network attacks were described in *the Interim Report #2* [IntRep#2]. They are briefly as follows:

(1) through inductive recovery based on the set of cases through the use of formal methods;
(2) by an expert who possesses knowledge of the malicious party's intentions and the possible ways these intentions can be realized;
(3) through combining the two above methods.

Two groups of *algorithms can be used for recovery of grammar specifying models of attacks*:
(1) enumeration grammar recovery algorithms;
(2) induction grammar recovery algorithms.

The *inductive grammar recovery methods* are deemed the most adequate for the purposes of recovering grammars that specify computer network attacks, specifically, the inductive method for recovering regular grammars on the basis of positive examples (the Feldman method). This method consists in constructing a non-recursive grammar that creates precisely those strings that were presented in the training sample, and then arriving at a simpler recursive grammar that generates all the strings of the positive examples and an infinite amount of other strings.

In order to demonstrate the performance capabilities of algorithms for recovery of grammars that specify different types of computer network attacks, we have looked at several cases of computer network attacks. These cases set the basic methods for implementing attacks of the following types: network scanning for identification of hosts; network scanning for identification of services; identification of operating system; shared resource enumeration; users and groups enumeration; applications and banners enumeration; actions on getting access to resources; denial of service attacks.

The *examples of using grammar recovery algorithms for specification of computer network attacks* were developed ([IntRep#2]). These examples showed practical applicability of the algorithms suggested for recovery of grammars specifying computer network attacks. The synthesized grammars can be used for generation of versions of attacks. The grammars developed through combining a number of productions are capable of generating the attacks that were not taken into account in the training cases. This expands the capability of the attack simulator that is based on the utilization of these grammars.

1.3. Mathematical methods and techniques realizing the attack formal modeling

Mathematical methods and techniques realizing the attack modeling were described in *the Interim Report #2* ([IntRep#2]) and defined more precisely in *the Interim Report #3* ([IntRep#3]).

We consider below the model of attack realization as a complex process of contest of adversary entities those are malefactor or team of malefactors, on the one hand, and network security system implementing a security policy, on the other hand.

1.3.1. Conceptual explanation of the attack modeling and simulation strategy

We defined the following *peculiarities of planning and execution of attacks*, influencing on choice of a formal model of attacks:

- *Any attack is target– and intention–centered,* i.e. it is directed against a particular object (network, computer, service, directory, file, etc.) and, as a rule, has a quite definite intention. *Intention* is understood as a goal or sub-goal a malefactor intends to achieve. We speak about malefactor's "intentions" according to the terminology used for mental concepts. Formally specified intention is called a "goal". *Examples of intentions*: reconnaissance (e.g. learning of network structure, identification of OS, hosts and/or services, etc.); penetration into the system; access to files of some directory; denial of service, etc. *Examples of targets*: *IP*-addresses of trusted hosts; password file; files of a particular directory; some resources of a particular host, etc. It should be noticed, that in some cases intention cannot be determined in advance. It can be accepted by malefactor in progress of attack development as a decision made on the basis of the obtained information and successfulness or ineffectiveness of particular malefactor's actions fulfilled earlier.
- *Attack intention can be represented in terms of partially ordered set of lower-level intentions.* A set of malefactor's intentions partially ordered in time is called an *attack scenario*. Intentions constituting attack scenario can be represented at different generalization levels. At the lowest level, each such intention is realized by a malefactor as a sequence of actions (network packets, commands of OS, etc.). Any malefactor's intention can be realized in multiple ways. Malefactor can vary the scenario implementing the same intention and the same attack object.
- *Attack modeling corresponds to an adversary domain.* Attack development depends on the result of each particular step of attack, i.e. it depends on response of the attacked network. In turn, a network response depends on security policy implemented. The current attack "state" is determined in terms of initial malefactor's information about the attacked network (or host),

information collected at preceding attack steps, and also the results (successfulness or ineffectiveness) of the preceding steps.

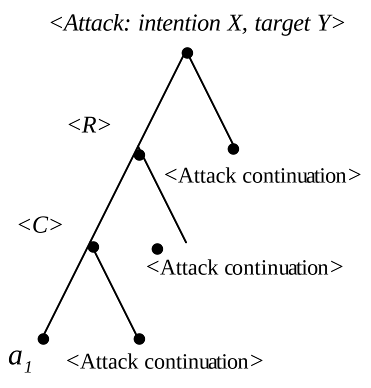


Fig.1.3.1. Attack scheme

Thus, any attack development depends on many random factors and, first of all, depends on attacked network response. Therefore, even if a general malefactor's intention is determined, *the attack development scenario cannot be definitely specified beforehand*.

An attack development depends on many uncertainties:

- uncertainty in choice of the attack intention and attack object;
- uncertainty caused by the information content with regard to the attacked network which a malefactor possesses at the beginning of attack and in progress of its development;
- uncertainty of choice of attack scenario implementing the already selected intention;
- uncertainty of the attacked computer network response.

The following <u>*scheme of attack generation (simulation)*</u> was developed.

Selection of the attack intention and attack object is a subjective act. Let the list $\boldsymbol{X}=\{X1, X_2,,\ldots, X_N\}$ of possible attack intentions and the list $\boldsymbol{Y}=\{Y_1, Y_2, \ldots, Y_M\}$ of attack objects be given. To select some attack intention and an attack object, it is necessary to set some formal mechanism of choice, for example, randomization mechanism. Let an intention $X\hat{I}\boldsymbol{X}$ and an attack object $Y\hat{I}\boldsymbol{Y}$ be selected.

The next component of attack modeling is a mechanism for generation of the attack given upper-level intention X and attack object Y in terms of hierarchy of lower-level malefactor's intentions and respective sequences of actions. Let us suppose that the malefactor's intention X consists in getting access to files in some directory of a host. If malefactor does not possess some basic information about computer network or host then he/she has to start from reconnaissance R, which corresponds to the first intention at the level that is lower with regard to the intention X of the top level. The reconnaissance R can be fulfilled, for example, by four different sub-attacks $\{A, B, C, D\}$. Only one of them can be selected on current step of the attack development as a sub-goal (intention) of the second level. We admit, that the malefactor has selected sub-goal C. Another malefactor in the same situation could make other selection. Therefore, it is quite reasonable to specify the above selection as a randomized step. Thus, generation of an attack in terms of lower level intentions given upper-level intention X and attack object Y can be formalized on the basis of randomization of choice among $\{A, B, C, D\}$.

Let the selected sub-goal C be a sequence of "commands", first of which be the command a_1. The term "command" is used here in the generalized sense. Main difference between "command" and "intention" consists in the following. The command is a concrete action; it is not a mental concept, which represents a certain abstraction in malefactor's mind. It can be a sequence of *IP*-packages, a command of operating system, etc. An intention is a component of the plan of actions; it is an "abstraction" represented formally at respective level of detail.

A set of sequences of commands, by which the malefactor tries to realize his /her intention, can be selected ambiguously. Therefore it is necessary to set a non-deterministic mechanism for generation of sequences of commands. It is obvious, that it can be randomization mechanism, however, probably, not so simple, as the random-number generator with a discrete distribution. Let a_1 be the first generated command. This command is dispatched to the attacked computer network (host). The hierarchy <attack intention X, attack target Y>→<lower level intentions>→<actions>, corresponding to the considered example scheme of the initial phase of the attack generation is shown in tree-like form in Fig.1.3.1.

The formally determined process of choice can be represented as follows: *<Attack: intention X, target Y> → <R> <Attack continuation, detailing X>, <R>→<C><Attack continuation, detailing R>, <C>→ a_1<Attack continuation, detailing C>*. The response of the attacked system to each command can be characterized as "success" if the command is executed like the malefactor wanted, or "failure", if the attacked system reacts to the command in the way that is undesirable for the malefactor. The next commands are determined in response of the attacked object to the command a_1.

If the chosen intention C is failed then the attack generation process can be stopped, or the attack can be continued starting with reselection of the choice associated with specialization of intention R – predecessor of the failed intention C in the tree (see Fig.1.3.1) in terms of the rest of the set $\{A, B, C, D\}$, i.e. in terms of one of the lower level intentions $\{A, B, D\}$. In the last case, the choice of a new alternative for the intention R specialization is made with the respective recalculation of the probability distribution given over the truncated set of lower-level intentions.

The next step and any subsequent one of attack generation is similar to the previous step. If in the following steps no one of intentions A, B and D does not result in success then the attack can be either finished or continued with the probable subsequent modification of the attack object. It is worth to notice, that in both above cases the probability distribution given over the set of the potentially admissible next step selections of intention alternatives should be recalculated.

If the attack with intention C is successful, then the attack can be stopped (if the goal is reached), or can be continued. This choice is also non-deterministic and can be simulated by a probabilistic mechanism and so on and so forth.

To an arbitrary step n of the attack generation (simulation) its state can be specified by a sequence of the following sort:

$$A(n) = \langle \textit{Attack prehistory} \rangle\ \langle \textit{Current state} \rangle\ \langle \textit{Attack continuation} \rangle,$$

where <*Attack prehistory*> is a sequence of the symbols corresponding to the preceding steps, in which each symbol is marked with a flag from a set {"success", "failure"}. This sequence can include symbols of intentions of different levels of detail, and symbols of actions. It is supposed, that the attack can be simulated at various levels of detail of the description; <*Current state*> is a partially unfolded sequence of the current attack step symbols; <*Attack continuation*> is still unknown part of the sequence $A(n)$, which generation is expected. In addition, current state of the attack development can also contain information collected at preceding steps.

It should be clear that it is impossible to enumerate and to specify all sequences $A(n)$, i.e. to specify completely in declarative form the total set of attacks and variants of their development mapped to total set variants of the attacked network responses. Therefore, the only way to specify attacks, if such way exists at all, is procedural way, which suppose to model attack by a generation algorithm. This way is used in our research.

While describing the developed model of attacks, we defined main notions of attack generation that are formalized in the problem domain ontology "*Computer network attacks*". In the developed formal model, the basic notions of the domain correspond to malefactor's intentions and all other notions are structured according to the structure of intentions. This is a reason why the developed approach is referred to as "*intention-centric approach*".

The following two *basic classes of high-lever malefactor's intentions* and their identifiers were used in the developed formal model:

- *R – Reconnaissance* aiming at getting information about the network (host). The particular cases of intentions of this class are *Identification of the running Hosts*, *Identification of the host Services*, *Identification of the host Operating system*, *Collection of additional Information about the network*, *Shared Resource Enumeration*, *Users and groups Enumeration*, *Applications and Banners Enumeration*.
- *I – Implantation and threat realization*. The particular cases of intentions of this class are *Getting Access to Resources of the host*, *Escalating Privilege with regard to the host resources*, *Gaining Additional Data needed for further threat realization*, *Threat Realization*, *Covering Tracks to avoid detection of malefactors' presence*, *Creating Back Doors*. Threat Realization can be detailed by the following sub-intentions: *Confidentiality destruction* (Confidentiality Violation Realization), for example, through getting access to file reading, *Integrity Destruction* (Integrity Violation Realization) realizing through attacks against integrity of the host resources, and *Denial of Service* (Availability Violation Realization).

The numbers, designations and interpretations of basic *malefactor's intentions* are explained in Tab.1.3.1.

Tab.1.3.1. The list of the malefactor's intentions

Number	Designation	Interpretation
1	*IH*	Identification of the running Hosts
2	*IS*	Identification of the host Services
3	*IO*	Identification of the host Operating system
4	*RE*	Resource Enumeration
5	*UE*	Users and groups Enumeration
6	*ABE*	Applications and Banners Enumeration
7	*GAR*	Gaining Access to Resources
8	*EP*	Escalating Privilege
9	*CVR*	Confidentiality Violation Realization or Confidentiality destruction
10	*IVR*	Integrity Violation Realization or Integrity Destruction
11	*AVR*	Availability Violation Realization or Denial of Service
12	*CBD*	Creating Back Doors

An attack *task* specification (or a *top-level attack goal*) can be specified by the following quad:

<Network (host) address, Malefactor's intention, Known data, Attack object>[2].

The task specification has to determine the class of scenarios that lead to the intended result. *Known data* specifies the information about attacked computer network (host) known for a malefactor. *Attack object* corresponds to the optional variable in attack goal specification. It is specified in the following ways:

- "**_**" – the attack object is not specified for the malefactor's intention "Reconnaissance" (*R*);
- If the intention corresponds to the attacks like *CVR* or *IVR* then the *attack object* is specified as follows: [*Account*,] [*Process* {*<Process name >/< Process mask >*},] [*File* {*<file name >/< file mask >*},] [*Data in transit* {*< file (data) name >/< file (data) mask >*}], where *Account* is object's account, *Process* is running process(es), *File* is file(s) that is the attack target(s) to get, *Data in transit* is data transmitting, where the variables in [] are optional, the repeatable variables are placed in {}, and symbol "/" is interpreted as "OR";
- "*All*" – all resources of the host (network);
- "*Anyone*" – at least one of the resources of the host (network).

1.3.2. Problem domain ontology: structure of the basic malefactors' intentions and actions

The developed problem domain ontology "*Computer network attacks*" comprises a hierarchy of notions specifying activities of malefactors directed to implementation of attacks of various classes in different levels of detail. In this ontology, the hierarchy of nodes representing notions splits into two subsets according to the *macro*- and *micro-levels* of the domain specifications. All nodes of the ontology of attacks at the macro- and micro-levels of specification are divided into the *intermediate* (can be further detailed) and *terminal* (cannot be detailed).

The notions of the ontology of an upper level can be interconnected with the corresponding notions of the lower level through one of three kinds of *relationships*:

- "*Part of*" that is the decomposition relationship ("*Whole*"–"*Part*");
- "*Kind of*" that is the specialization relationship ("*Notion*"–"*Particular kind of notion*");
- "*Seq of*" that is the relationship specifying sequence of operation ("*Whole operation*" – "*Sub-operation*").

High-level notions corresponding to the intentions form the upper levels of the ontology. They are interconnected with the "*Part of*" relationship. Attack actions realizing malefactor's intentions are interconnected with the intentions by "*Kind of*" or "*Seq of*" relationship. The developed ontology

[2] In the software tool this 4-tuple is used for specification of simulation task by user.

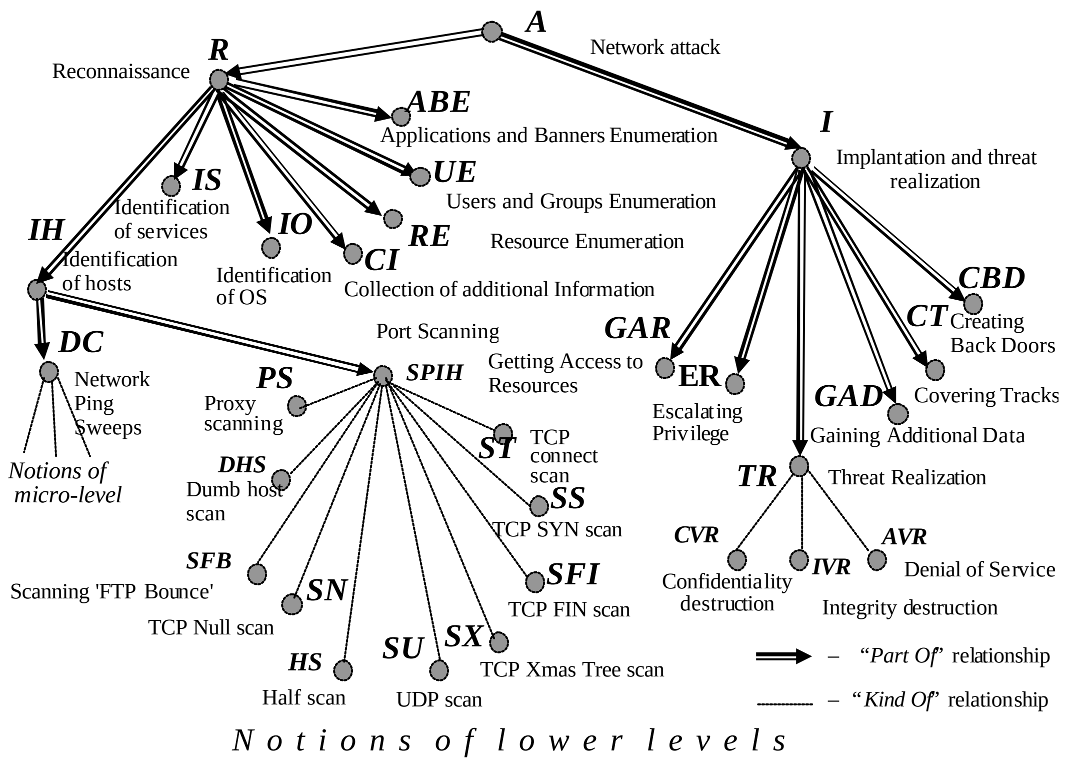


Fig.1.3.2. Macro-level fragment of the domain ontology "*Computer network attacks*"

includes detailed descriptions of the network attack domain in which the notions of the bottom level ("*terminals*") can be specified in terms of network packets, OS calls, and audit data.

Let us look at a high-level fragment of the developed ontology (Fig.1.3.2). At the upper-level of the *macro-specification of attacks,* the notion of "Network Attack" (designated by *A*) is in the "*Part of*" relationship to the "Reconnaissance" (*R*) and "Implantation and threat realization" (*I*). In turn, the notion *R* is in the "*Part of*" relationship to the notions *IH*, *IS*, *IO*, *CI*, *RE*, *UE*, and *ABE*. The notion *I* is in the "*Part of*" relationship to the notions *GAR*, *EP*, *GAD*, *TR*, *CT*, and *CBD*. In the next (lower) level of the hierarchy of the problem domain ontology, for example, the notion *IH* is in the "*Kind of*" relationship to the notions "Network Ping Sweeps" (*DC*) and "Port Scanning" (*SPIH*). At that, the notion "Network Ping Sweeps" (*DC*) is the lowest ("terminal") notion of the macro-level of attack specification, and the notion "Port Scanning" (*SPIH*) is detailed through the use of the "*Kind of*" relationship by a set of "terminal" notions of the macro-level of attack specification.

The "terminal" notions of the macro-level are further detailed at the *micro-level of attack specification***,** and on this level they belong to the set of top-level notions detailed through the use of the three relationships introduced above. Thus, for example, the notion "Network Ping Sweeps" (*DC*) is in the "*Kind of*" relationship with the notions "Network Ping Sweeps with *ping*" (*PI*), "Network Ping Sweeps with *Ping Sweep*" (*PSW*), etc., which, in turn, correspond to the names of utilities that perform "Network Ping Sweeps".

In turn, each of these notions, e.g. "Network Ping Sweeps with *Ping Sweep*" (*PSW*), is in the "*Seq of*" relationship to the "ICMP ECHO REQUEST" (*IER*) notions. The "ICMP ECHO REQUEST" (*IER*) notions correspond to network packets that are directed at the host (or the network) – the target of the attack.

In micro specifications of the attacks ontology, besides the three relations described ("*Part of*", "*Kind of*", "*Seq of*"), the relationship "*Example of*" is also used. It serves to establish the "type of

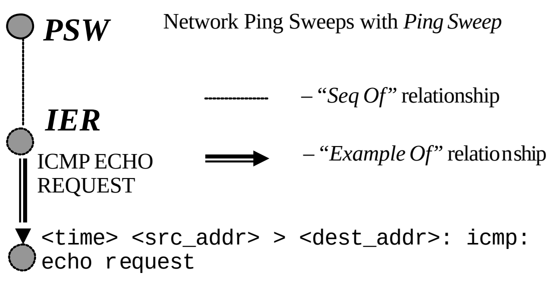

PSW
Network Ping Sweeps with Ping Sweep
– "Seq Of" relationship
IER
ICMP ECHO REQUEST
– "Example Of" relationship
<time> <src_addr> > <dest_addr>: icmp: echo request

Fig.1.3.3. Micro-level fragment of the domain ontology "*Computer network attacks*"

object – specific sample of object" relationship. In Fig.1.3.3, this type of relationship is used to establish the connection between the echo-request of the protocol ICMP ("ICMP ECHO REQUEST") and its specific implementation specified, for example, as a message <time> <src_addr> > <dest_addr>: icmp: echo request, where <time> – time stamp, <src_addr> – source IP address, <src_port> – source port, <dest_addr> – destination IP address.

1.3.3. Formal grammar framework for specification of computer network attacks

Being based on explanation of the attack modeling strategy, definition of basic notions of attack specification, structure of the basic malefactors' intentions and actions, the following basic assumptions and statements used for formal attack specification were determined ([IntRep#2], [IntRep#3]):

- Each attack intention can be considered as a *sequence of symbols* in terms of lower-level intentions and actions. These sequences can be formally considered as "words" of a language, which can be generated by a formal grammar. Thus, each node of the ontology *"Computer network attacks"* can be specified in terms of a formal grammar generating more detailed attack specification;
- Analysis of a wide spectrum of formal grammar-based specifications of attack intentions justified that attack intentions can be adequately specified in terms of *LL(2)* context-free grammars;
- Specification of uncertainties inherent to the attack development can be done in probabilistic terms through attributes and functions given over them. Thus, the resulting framework for attack specification can be restricted to a stochastic attribute grammar;
- Each node (grammar) of the ontology is interconnected with the upper level node (grammar) and this interconnection can be specified through "grammar substitution" operation [Glushkov *et al*-78] in which a terminal symbol of the parent node is considered as the axiom of the grammar corresponding to its child node;
- Each malefactor's action has to be followed by an attacked network response.

Thus, mathematical model of attack intentions was determined in terms of a set of *formal grammars* specifying particular intentions interconnected through "*substitution*" operations: $M_A=<\{G_i\}, \{Su\}>$, where $\{G_i\}$ – the formal grammars, $\{Su\}$ – the "substitution" operations.

Every formal grammar is specified by quintuple

$$G = < V_N, V_T, S, P, A >,$$

where G is the grammar name, V_N is the set of non-terminal symbols (that are associated with the upper and the intermediate levels of an attack scenario), V_T is the set of its terminal symbols (that designate the malefactors' actions represented as steps of a lower-level attack scenario), $S \hat{I} V_N$ is the grammar axiom (an initial symbol of an attack scenario), P is the set of productions that specify the specialization operations for the intention through the substitution of the symbols of an upper-level node by the symbols of the lower-level nodes, and A is the set of attributes and algorithms of their computation.

Attribute component of each grammar serves for two main purposes:

- The first of them is to specify *randomized choice of a production* at the current inference step if several productions have the equal left part non-terminals coinciding with the "active" non-terminal in the current sequence under inference. These probabilities are recalculated on-line subject to the prehistory of attack development and previous results of attack. So, in order to specify a stochastic grammar, each production is supplemented with a specification of the probability of the rule being chosen in the inference process.
- Also the attribute component is used to check *conditions determining the admissibility of using a production* at the current step of inference. These conditions depend on attack task specification, attacked computer network (host) response and also on the malefactor's previous actions. These conditions may depend on compatibility of malefactor's actions and attacked network or host properties, e.g., OS type and version, running services, security parameters, etc.

These are the examples of host parameters, which may form production conditions: (1) OS types – *Unix, Linux, Win* (all Windows OS), *9x* (*95, 98, Me*), *NT* (*NT, 2000*), *SunOS, Solaris*, etc.; (2) running applications – e.g., *PWS* – an initial version of Microsoft's Personal Web Server is running; (3) protection parameter – *CFP* (shared files and printers), *NS* (Null Sessions), *PA* (Password is Absent), *RR* (Remote Registry), etc.; (4) additional parameters – *AS* (Access to Segment of LAN), *THD* (Trusted Host Data), etc.

If it is necessary to specify several parameters, operations "OR" (signified by ",") and (or) "AND" (".") are used. Relationships of ownership and membership are also taken into account, e.g. $SunOS \in Unix$; $\{95, 98, Me\} \subset 9x$; $\{95, 98, Me, NT, 2000, XP\} \subset Win$, $9x \in Win$, etc.

Thus, in general case, the grammar production is recorded as follows: $[(U)]\ X ® \ a\ (Prob)$, where U – the condition for the rule usage, [ ] – an optional element, X – non-terminal symbol, a –a string of terminal and non-terminal symbols, *Prob* – the initial value of probability of the rule usage.

Let us explain by example the operation of *grammar substitution* and its role in the formal model of attacks. Let $a\,Î\,V_T(G_i)$ be a terminal symbol of the grammar G_i in the sequence of symbols generated by the grammar G_i. Symbol a denotes the name of a particular intention or attack action, and $G(a)$ is the grammar generating variants of the a implementation. Then, *operation* $Su(a)$ *of substitution* $G(a)$ in place of symbol a is specified in the form $Su(a)$: $\{a ® G(a)\}$. *Semantics of this operation* is that in place of symbol a in already generated sequence any "word" generated by grammar $G(a)$ can be placed. In fact, this operation corresponds to a step towards the more detailed specification of an attack scenario.

When the micro specifications are used for modeling of attacks, it is necessary to use the ontology nodes of the lowest (terminal) level and substitute specific values for the variables that determine the attack task specification. For example, let us suppose a ping attack is being implemented using "Network Ping Sweeps with *Ping Sweep*" (*PSW*). *PSW* is in the "*Seq of*" relationship to the "ICMP ECHO REQUEST" (*IER*) network packets that are directed at the target host (network). In micro specifications of attacks the *IER* node is in the "*Example of*" relationship to its specific implementation defined as the following message: <time> <src_addr> > <dest_addr>: icmp: echo request ,where <time> – time stamp, <src_addr> – source IP address, <src_port> – source port, <dest_addr> – destination IP address. The grammar that specifies *PSW* may look like this: V_N={PSW, PSW1}, V_T={IER}, S={PSW}, P={PSW $\to$ IER PSW1 (1), PSW1 $\to$ IER PSW1 (0.2), PSW1 $\to$ IER (0.8)}.

Let us suppose a ping attack with "*Ping Sweep*" is being implemented from host 244.146.4.20 on the hosts of the network 198.24.15.0 in the time interval [0:43:10.094644, 00:43:16.036735]. Let us suppose that the string "*IER IER*" was created as a result of using the *PSW* grammar. Then, based on the "*Example of*" relationship, the symbols of this string should generate two messages:

<time1> <src_addr> > <dest_addr>: icmp: echo request ,

<time2> <src_addr> > <dest_addr>: icmp: echo request .

After the parameterization <time1> = 00:43:10.094644, <src_addr>=244.146.4.20, <dest_addr> = 198.24.15.255, <time2>=00:43:16.036735, these messages should look like these:

00:43:10.094644 244.146.4.20>198.24.15.255:icmp:echo request and 00:43:16.036735 244.146.4.20>198.24.15.255:icmp:echo request,

which correspond to the *icmp*-packets sent to the network hosts 198.24.15.0 (since the X.X.X.255 address is specified in the *icmp*-packets, the packets are sent to all the hosts of the specified networks).

1.3.4. Formal models of a representative multitude of computer network attacks

The development of the family of grammars $\{G_i\}$ was conducted in the following order ([IntRep#2], [IntRep#3]):

- First, for each basic malefactor's intention, its own family of enclosed attributed stochastic context-free grammars was constructed;
- Second, these families of grammars were transformed into the generalized grammars that correspond to each non-terminal node of ontology for all of the intentions.

It is assumed that if a value of the production condition is not determined at the moment of production selection all available productions may be used at the respective step of attack simulation. Also it is supposed that the terminal actions generated by productions are associated with the probabilities of successful realization of those actions (attacks) and the host response.

Each *grammar corresponding to basic malefactor's intention* is described through four elements: (1) the set of non-terminal symbols V_N, (2) the set of terminal symbols V_T, (3) the first symbol S and (4) the set of productions P.

Each production is established as:

[(U)] X ® a (Prob),

where *U* – the condition for usage of the rule, *[ ]* – indicates that the element if braces is optional, *X* – non-terminal symbol, *a* –the string of terminal and non-terminal symbols, *Prob* – the initial value of the probability of the rule being chosen for given intention.

Let us consider two *examples of developed grammars for basic malefactor's intentions*: (1) grammars for the intention "Identification of Operating system" (IO) and (2) grammars for the intention "Users and groups Enumeration" (UE).

The *family of attributed stochastic context-free grammars for the intention "Identification of Operating system" (IO)* can be represented as follows:

Grammar "Network Attack":

V_N={A, A1}, V_T={R}, S={A}, P={A ® A1 (1), A1 ® R (0.7), A1 ® R A1 (0.3)}.

Grammar "Reconnaissance":

V_N={R, R1}, V_T={IO}, S={R}, P={R ® R1 (1), R1 ® IO (0.7), R1 ® IO R1 (0.3)}.

Grammar "Identification of Operating system":

V_N={IO, IO1},
V_T={TZ, TS, FF, RF, RS, II, IL, MD, IW, MA, IV, IF, IP, ISP, IDOS}, S={IO},
P={IO ® IO1 (1), IO1 ® TZ (0.05), (Unix, Linux) IO1 ® TS (0.05),
(Unix, Linux) IO1 ® FF (0.05), IO1 ® RF (0.05), IO1 ® RS (0.05),
IO1 ® II (0.05), IO1 ® IL (0.05), IO1 ® MD (0.05),
IO1 ® IW (0.05), IO1 ® MA (0.05), IO1 ® IV (0.04),
IO1 ® IF (0.04), IO1 ® IP (0.04),
IO1 ® IS (0.04), (Win) IO1 ® IDOS (0.04),
IO1 ® TZ IO1 (0.02), (Unix, Linux) IO1 ® TS IO1 (0.02),
(Unix, Linux) IO1 ® FF IO1 (0.02), IO1 ® RF IO1 (0.02),
IO1 ® RS IO1 (0.02), IO1 ® II IO1 (0.02), IO1 ® IL IO1 (0.02),
IO1 ® MD IO1 (0.02), IO1 ® IW IO1 (0.02), IO1 ® MA IO1 (0.02),
IO1 ® IV IO1 (0.02), IO1 ® IF IO1 (0.02),
IO1 ® IP IO1 (0.02), IO1 ® IS IO1 (0.02),
(Win) IO1 ® IDOS IO1 (0.02)}.

In this set of grammars the following denotations are used: *A* – Network Attack; *R* – Reconnaissance; *IO* – Identification of Operating system; *TZ* – Connection on telnet and examination of the message header about operating system; *TS* – Connection on telnet and execution of the SYST command; *FF* – Connection on FTP and examination of bin-files in the directory /bin/ls; *RF* – FIN Probe - Exploration by the FIN package; *RS* – Bogus flag Probe - Exploration by the package SYN with a false (unused) flag (BOGUS-flag); *II* – ISN sampling - Capture of initial sequential number ISN at response to a TCP SYN connection request; *IL* – Definition of the law of the ISN change; *MD* – Monitoring of the fragmentation prohibition bit DF; *IW* – Watching of an initial size of the TCP window; *MA* – Watching of value of sequential number used for a field ACK; *IV* – Watching of an initial size of the TCP window; *IF* – FIN Probe - Exploration by the FIN package; *IP* – Examination of the answer for sending of the TCP packet with certain values of a field "Options"; *ISP* – Infecting

Startup Files; *IDOS* – Examination of response for DoS attacks Ping of Death, WinNuke, Teardrop, Land for detection of a Windows OS type; *A1, R1, IO1*– auxiliary symbols.

The *family of attributed stochastic context-free grammars for the intention "Users and groups Enumeration" (UE)* can be described as follows:

Grammar "Network Attack":

V_N*={A, A1}*, V_T*={R}, S={A}, P={A ® A1 (1), A1 ® R (0.7), A1 ® R A1 (0.3)}.*

Grammar "Reconnaissance":

V_N*={R, R1}*, V_T*={UE}, S={R}, P={R ® R1 (1), R1 ® UE (0.7), R1 ® UE R1 (0.3)}.*

Grammar of the level "Users and groups Enumeration":

V_N*={UE, UE1, UE2, UE3, UE4}, S={UE},*
V_T*={DNNT, EUE, PIUD, IAUS, SNMPE, FUE, UTFTP},*
$P_{for\ Windows\ 9x,Me,NT,2000}$*={(Win) UE® UE1(1), (NS)UE1 ® UE2(0.65), UE1 ® SNMPE(0.25), UE1®SNMPE UE1 (0.05), (NS)UE1 ®UE2 UE1 (0.05), (&)UE2®CNS UE3(1), UE3 ®DNNT (0.2), UE3 ®DNNT UE4 (0.05), UE3 ®IAUS(0.35), UE3 ®EUE(0.2), UE3 ® PIUD (0.2), UE4®DNNT UE4(0.1), UE4®DNNT(0.9)},*
$P_{for\ Unix/Linux}$ *={(Unix, Linux) UE®UE1(1), UE1 ®FUE(0.3), UE1®SNMPE(0.2), UE1®UTFTP(0.1), UE1®FUE UE1(0.1), UE1®SNMPE UE1(0.1), UE1®UTFTP UE1(0.2)}.*

Grammar "Identifying Accounts with user2sid/sid2user":

V_N*={IAUS, IAUS1, IAUS2}*, V_T*={ISU, IAS}, S={IAUS},*
P={(NT) IAUS ® IAUS1 (1), (&) IAUS1 ® ISU IAS (0.8), IAUS1 ® IAUS1 IAUS2 (0.2), (&) IAUS2 ® ISU IAS (1)}.

In the second set of grammars the following denotations are used: *A* – Network Attack; *R* – Reconnaissance; *UE* – Users and groups Enumeration; *DNNT* – Dumping the NetBIOS Name Table with *nbtstat* and *nbtscan*; *EUE* – Enumerating Users with *enum*; *PIUD* – Providing Information about Users with *DumpSec* (*DumpACL*); *IAUS* – Identifying Accounts with *user2sid/sid2user*; *SNMPE* – SNMP Enumeration with *snmputil* or *IP Network Browser*; *FUE* – Finger Users Enumeration; *UTFTP* – Use of Trivial File Transfer Protocol for Unix enumerating by stealing */etc/passwd* and (or) */etc/hosts.equiv* and (or) *~/.rhosts*; *ISU* – Identifying SID with *user2sid*; *IAS* – Identifying Account with *sid2user* using user's RID; *A1, R1, UE1, UE2, UE3, UE4, IAUS1, IAUS2* – auxiliary symbols.

Each *generalized grammars* were described through four elements: V_N, V_T, *S*, and *P*. Specifications of these elements are analogous to grammars corresponding basic malefactor's intentions with the exception of the descriptions of productions. For each production *[(U)] X ® a (Prob)*, instead of the element *(Prob)*, the set of probabilities of the choice of the rule for each i^{-th} intention is set as follows: S_i*= Prob*. If $S_i=0$, the corresponding probability is not assigned. The generalized grammars are stochastic attributive *LL(2)*-grammars. *LL(2)* means that the strings of such a grammar are generated left to right, top to bottom with uncertainty of the choice of substitution through the second symbol.

As an *example of the generalized grammars* let us consider the grammar *"Getting Access to Resources"* (*GAR*):

V_N*={GAR, GAR1}*, V_T*={DCSR, IBSD, EKV, CPF, AAF, BFPG, PSA, CSS, End}, S={GAR},*
P={GAR ® DCSR GAR1 ($S_7=0.2, S_8=0.2, S_9=0.2, S_{10}=0.2, S_{11}=0.2, S_{12}=0.2$*),*
GAR ® IBSD GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$*),*
GAR ® EKV GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$*),*
GAR ® CPF GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$*),*
GAR ® AAF GAR1 ($S_7=0.2, S_8=0.2, S_9=0.2, S_{10}=0.2, S_{11}=0.2, S_{12}=0.2$*),*

GAR ® BFPG GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
GAR ® PSA GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
GAR ® CSS GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Win) GAR ® DCSR GAR1 ($S_7=0.3, S_8=0.3, S_9=0.3, S_{10}=0.3, S_{11}=0.3, S_{12}=0.3$),
(Win) GAR ® IBSD GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Win) GAR ® EKV GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Win) GAR ® CPF GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Win) GAR ® AAF GAR1 ($S_7=0.2, S_8=0.2, S_9=0.2, S_{10}=0.2, S_{11}=0.2, S_{12}=0.2$),
(Win) GAR ® BFPG GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Win) GAR ® PSA GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Windows 9x) GAR ® DCSR GAR1 ($S_7=0.4, S_8=0.4, S_9=0.4, S_{10}=0.4, S_{11}=0.4, S_{12}=0.4$),
(Windows 9x) GAR ® IBSD GAR1 ($S_7=0.3, S_8=0.3, S_9=0.3, S_{10}=0.3, S_{11}=0.3, S_{12}=0.3$),
(Windows 9x) GAR ® EKV GAR1 ($S_7=0.2, S_8=0.2, S_9=0.2, S_{10}=0.2, S_{11}=0.2, S_{12}=0.2$),
(Windows 9x) GAR ® CPF GAR1 ($S_7=0.1, S_8=0.1, S_9=0.1, S_{10}=0.1, S_{11}=0.1, S_{12}=0.1$),
(Windows NT) GAR ® AAF GAR1 ($S_7=0.4, S_8=0.4, S_9=0.4, S_{10}=0.4, S_{11}=0.4, S_{12}=0.4$),
(Windows NT) GAR ® BFPG GAR1 ($S_7=0.3, S_8=0.3, S_9=0.3, S_{10}=0.3, S_{11}=0.3, S_{12}=0.3$),
(Windows NT) GAR ® PSA GAR1 ($S_7=0.3, S_8=0.3, S_9=0.3, S_{10}=0.3, S_{11}=0.3, S_{12}=0.3$),
(Unix) GAR ® AAF GAR1 ($S_7=0.25, S_8=0.25, S_9=0.25, S_{10}=0.25, S_{11}=0.25, S_{12}=0.25$),
(Unix) GAR ® BFPG GAR1 ($S_7=0.25, S_8=0.25, S_9=0.25, S_{10}=0.25, S_{11}=0.25, S_{12}=0.25$),
(Unix) GAR ® PSA GAR1 ($S_7=0.25, S_8=0.25, S_9=0.25, S_{10}=0.25, S_{11}=0.25, S_{12}=0.25$),
(Unix) GAR ® CSS GAR1 ($S_7=0.25, S_8=0.25, S_9=0.25, S_{10}=0.25, S_{11}=0.25, S_{12}=0.25$),
(SunOS 1.4.x, THD) GAR ® CSS GAR1 ($S_7=1, S_8=1, S_9=1, S_{10}=1, S_{11}=1, S_{12}=1$),
GAR1 ® End ($S_7=1, S_8=1, S_9=1, S_{10}=1, S_{11}=1, S_{12}=1$)}.

In this grammar the following denotations are used: *GAR* – Getting Access to Resources; *DCSR* – Direct Connection to a Shared Recourse; *IBSD* – Installation of Backdoor Server Daemons and Trojans and access to a host; *EKV* – Exploitation of Known server application Vulnerabilities; *CPF* – Cracking of PWL File and access to a host; *AAF* – Anonymity Access to Ftp-server; *BFPG* – Brute Force Password Guessing and access to a host; *PSA* – Password Stealing Attack and access to a host; *CSS* – Combined IP spoofing on SunOS v.1.4.x; *GAR1*, *End* – auxiliary symbols.

1.3.5. State machine-based implementation of the attack generation

Algorithmic representation of the attack generation specified as a family of formal generalized grammars was implemented by a family of state machines ([IntRep#2], [IntRep#3]).

The basic elements of each state machine are states, transition arcs, and explanatory texts for each transition. States of each state machine are divided into three types:

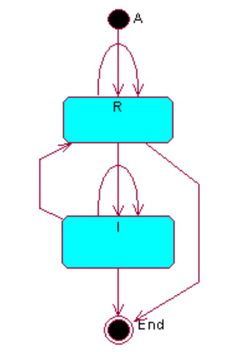
- first (initial),
- intermediate, and
- final (marker of this state is *End*).

The initial and intermediate states are the following:

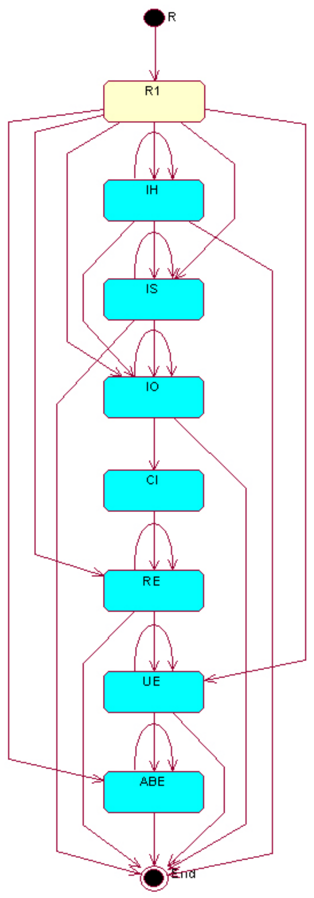
- non-terminal, those that initiate the work of the corresponding nested state machines;
- terminal, those that interact with the host model;
- abstract (auxiliary) states.

Transition arcs are identified with the productions of grammars, and can be carried out only under certain conditions. Within the state, besides the transition choice depending on the intention and the current transition probability, the following types of action can be performed:

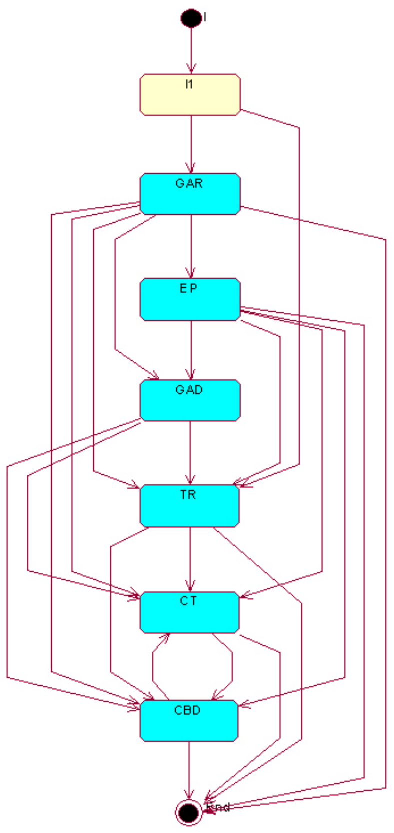
- *Entry action* (an action performed at entering the state);
- *Do action* (a set of basic actions, including actions of transition to the nested state machine or realizing the host response model);
- *Exit action* (an action performed at exit).



A – Network Attack; *R* – Reconnaissance; *I* – Implantation and Thread Realization; *End* – final state

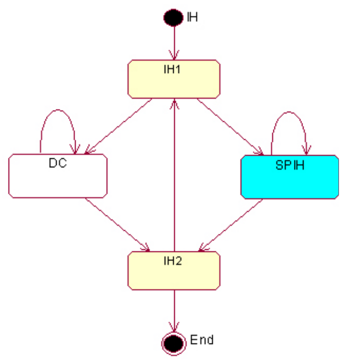


R – Reconnaissance; *R1* – Intermediate state; *IH* – Identification of the running Hosts; *IS* – Identification of the host Services; *IO* – Identification of the host OS; *CI* – Collection of Information; *RE* – Resource Enumeration; *UE* – Users and Groups Enumeration; *ABE* – Applications and Banners Enumeration; End – final state

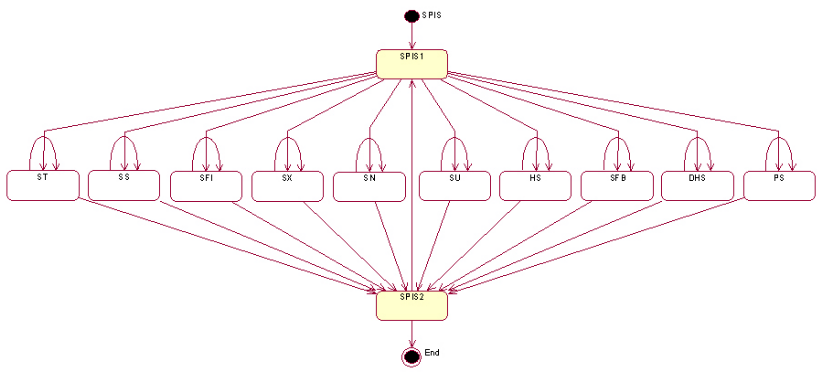


I - Implantation and Thread Realization; I1 – Intermediate state; *GAR* – Getting Access to Resources; *EP* – Escalating Privilege; *GAD* – Gaining Additional Data; *TR* – Thread Realization; *CT* – Covering Tracks; *CBD* – Creating Back Doors; End – final state

(a) Network Attack (b) Reconnaissance (c) Implantation and threat realization



IH – Identification of the running Hosts; *IH1*, *IH2* – Intermediate states; *DC* – Network Ping Sweeps; *SPIH* – Port Scanning; *End* – final state



SPIS – Port Scanning during Identification of Services; *SPIS1*, *SPIS2* – Intermediate states; *ST* – TCP connect scan; *SS* – TCP SYN scan; *SFI* – TCP FIN scan; *SX* – TCP Xmas Tree scan; *SN* – TCP Null scan; *SU* – UDP scan; *HS* – Half scan; *SFB* – Scanning “FTP Bounce”; *DHS* – Dumb host scan; *PS* – “Proxy”- scanning End – final state

(d) Identification of Hosts (e) Port Scanning during Identification of Services

Fig.1.3.4. Examples of state machine diagrams

The model of each state machine was set by specifying the following components:

- diagram of state machine;
- main parameters of the state machine;
- parameters of transitions that determine the stochastic model of the state machine functioning for different relevant intentions regarding the implementation of network attacks;
- transition conditions.

In the *state machine diagram*, the first and the final states are signified by black circles, and the intermediate states – by rectangles with rounded corners.

Arrows signify the transitions of the state machine. Dotted lines connect the transition arcs to explanatory texts with the grammar rules.

The explanatory texts contain number and contents of *grammar production* that corresponds to the transition, numbers of *intentions* for which the rule should be implemented (numbers of intentions are in parenthesis), *conditions* of the transition (if they exist).

The *main parameters of the state machine* includes state machine name, relevant intentions, states, first state, non-terminal, terminal and auxiliary states.

The parameters of the transitions from every state are described in a table. Several lines of this table can be assigned to each state; their number is determined by the number of transitions from this state.

Each line of the table includes:

N – the number of the transition rule; *CS* – current state; *Rule* – the corresponding production of the grammar; *NS* – the state achieved through the application of this rule;

Cond – the marker of the transition condition; probabilities P_i of the choice of the transition depending on the realization of the i^{th} intention and factor *K* (in the gray sub-string), the multiplier for the current probability *Pi* in case of recursive application of the rule aimed at limiting the length of the generated string of action symbols. After the multiplication by *K*, all the other rules of transition from this state are normalized. The probability of choice of each rule is divided by the sum of probabilities of all rules of transition from this state. The resulting sum of all probabilities equals to 1;

Actions (*Entry* action, *Do* action, *Exit* action) are actions that are to be performed upon the transition to the next state. Actions may be absent.

Each rule is applied under the condition that the intention is within the permissible range of intentions of the rule. Gray is used to designate the sub-string that contains the values for the factor *K*.

Examples of the state machines diagrams “Network Attack” (*A*), “Reconnaissance” (*R*), “Implantation and threat realization” (*I*), “Identification of Hosts” (*IH*) and “Port Scanning during Identification of Services” (*SPIS*) without explanatory texts are represented in Fig.1.3.4.

1.3.6. Formal model of the attacked computer network and its response to attacks

The attack development depends on the malefactor's “skill”, information regarding network characteristics, which he/she possesses, some other malefactor's attributes, security policy of the attacked network, etc. An attack proceeds as interactive process, in which the network reacts to the malefactor's action. Computer network plays the role of the environment for attacker, and therefore its model must be a part of the attack simulation tool.

The peculiarity of any attack is that the malefactor's strategy depends on the results of the intermediate actions. This is the reason why it is not possible to generate the complete sequence of malefactor's actions from the very beginning. The malefactor's action has to be generated on-line in parallel with the getting reaction from the attacked network. The proposed context-free grammar syntax provides the model with this capability. At each particular step of inference, it generates no more than single terminal symbol that can be interpreted by the computer network model as a malefactor's action. The network returns the value of the result (success or failure). The model of attacker receives it and generates the next terminal symbol according to the attack model and depending on the returned result of the previous phase of the attack.

Model of the attacked computer network and its response to attacks is represented as the following quadruple ([IntRep#2], [IntRep#3]):

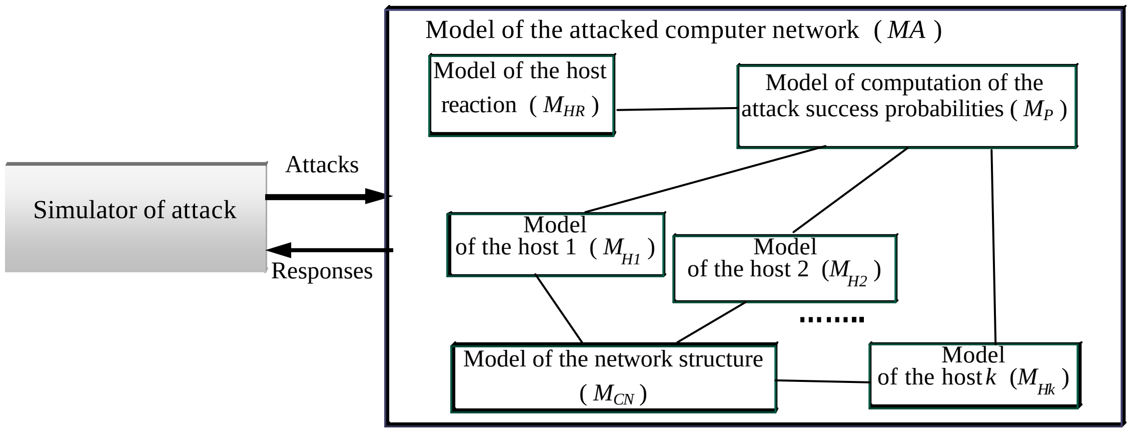
$$MA = \langle M_{CN}, \{M_{H_i}\}, M_P, M_{HR} \rangle,$$

where M_{CN} – the model of the network structure; $\{M_{Hi}\}$ – a set of models of the host resources; M_P – the model of computation of attack success probabilities; M_{HR} – the model of the host reaction in response of attack. The diagram of interaction of these sub-models under effect of the attack simulator is represented in Fig.1.3.5.

The *model M_{CN} of the computer network structure* was determined as follows:

$$M_{CN} = < A, P, N, C>,$$

where *CN* – the computer network identifier, *A* – the network address; *P* – a family of protocols used (e.g., TCP/IP, FDDI, ATM, IPX, etc.); *N* – a set $\{CN_i\}$ of sub-networks and/or a set $\{H_i\}$ of hosts of the network *CN*; *C* is a set of connections between the sub-networks (hosts) established as a mapping matrix. If *N* establishes a set of sub-networks $\{CN_i\}$, then each sub-network CN_i can in turn be specified by the model M_{CNi} (if its structure needs to be developed in detail and if information is available about this structure). Each host H_i is determined as a pair M_{Hi}= <*A, T*>, where *A* is the host address, *T* is a host type (e.g., firewall, router, host, etc.).



Model of the attacked computer network (MA)
Model of the host reaction (M_{HR})
Model of computation of the attack success probabilities (M_P)
Simulator of attack
Attacks
Responses
Model of the host 1 (M_{H1})
Model of the host 2 (M_{H2})
....
Model of the network structure (M_{CN})
Model of the host k (M_{Hk})

Fig.1.3.5. Interaction of the attacked computer network sub-models under effect of the attack simulator

The *models $\{M_{Hi}\}$ of the network host (resources)* serve for representing the host parameters that are important for attack simulation. The model of the network host resources is determined as follows:

$$M_{Hi} = < A, M, T, N, D, P, S, DP, ASP, RA, SP, SR, TH, \text{etc.}>,$$

where *A* – IP-address, *M* – mask of the network address, *T* – type and version of OS, *N* – users' identifiers (IDs), *D* – domain names, *P* – host access passwords, *S* – users' security identifiers (SID), *DP* – domain parameters (domain, names of hosts in the domain, domain controller, related domains), *ASP* – active TCP and UDP ports and services of the hosts, *RA* – running applications, *SP* – security parameters, *SR* – shared resources, *TH* – trusted hosts.

Success or failure of any attack action (corresponding to terminal level of the attack ontology) is determined by means of the *model M_P of computation of the attack success probabilities*. This model was specified as follows:

$$M_P = \{R^{SPr}_j\},$$

where R^{SPr}_j is a special rule that determines the action success probability depending on the basic parameters of the host (attack target). The rule R^{SPr}_j includes *IF* and *THEN* parts. The *IF* part contains action name and precondition (values of attributes constraining the attack applicability). The *THEN* part contains the value of success probability (*SPr*). Examples of interpretations of the probability computation rules are as follows:

"If action is 'FF' (Connection on FTP and examination of bin-files in the directory /bin/ls) and OS Type is 'Unix, Linux' and Service is 'FTP' then SP is 0.7";

“If action is ‘FCA’ (Free Common Access) and OS type is ‘Windows 9x’ and Security parameter is ‘CFP’ (shared files and printers) then SP is 0.7”.

The result of each attack action is determined according to the *model M_{HR} of the host reaction*. This model is determined as a set of rules of the host reaction:

$$M_{HR} = \{R^{HR}_j\},$$
$$R^{HR}_j: \textit{Input} \rightarrow \textit{Output}\ [\&\ \textit{Post-Condition}],$$

where *Input* – the malefactor’s activity, *Output* – the host reaction, *Post-Condition* – a change of the host state, & – logic connective *“AND*”, *[ ]* – optional part of the rule.

The *Input* format:

<Attack name>: <Input message> : <Attack objects> [; *<Objects involved in the attack>*].

The *Output* format:

{*<Attack success parameter S> [: <Output message>];*
{*<Attack success parameter F> [: <Output message>]*}.

The Attack Success Parameter is determined by the success probability of the attack that is associated with the host (attack target) depending on the implemented attack type. The values of attack success parameter are *Success* (*S*), and *Failure* (*F*).

The part of output message shown in the < > is taken from the corresponding field of the host (target) parameters. The part of output message shown in quotation marks “ ” is displayed as a constant line.

The *Post-Condition* format:

$$\{p_1=P_1, p_2=P_2,, \ldots, p_n=P_n\},$$

where p_i – ith parameter of the host (for instance, *SP, SR, TH,* etc.) which value has changed, P_i – the value of ith parameter.

Examples of the host reaction rules:

SFB: Scanning “FTP Bounce” : Target host; Intermediate host (FTP-server) → {*S: <Active ports (services) of a host>; F: “It was not possible to determine Active ports (services)”*};

IF: ICMP message quoting : Target host → { *S: <The type of operating system>; F: “It was not possible to determine the type of operating system”*}.

1.4. Object-oriented project of the Attack Simulator–software tool prototype for simulation of attacks on the computer network

The object-oriented project of the Attack Simulator prototype is described in detail in *the Interim Report #3* [IntRep#3].

1.4.1. Peculiarities of the developed technology for Attack Simulator design

For development of the object-oriented project of the Attack Simulator and its implementation the special technology and software tool – the Multi-agent System Development Kit (MAS DK) designed for design and implementation of multi-agent systems was used.

Till now a good deal of such software tools for design and implementation of multi-agent systems (MAS) have been developed. Between them, the most known ones are such as AgentBuilder [AgentBuilder-99], MadKit [Madkit], Bee-gent [Bee-gent-00], FIPA-OS [Poslad *et al*-00], JADE [Bellifemine *et al*-99], Zeus [Collis *et al*-99], etc.

However, they do not meet wholly the present-day requirements [Sloman-00].

The first group of the requirements corresponds to the properties of the technology provided by a software tool, including support for the entire circle of MAS design, friendly interfaces for all categories of users, visual programming style, concurrency of the development, automated documenting, etc.

The second group concerns the target MAS properties (ontology, inference mechanisms, behavior scenarios, communication component, etc.). These requirements form the focus of MAS DK being developed by authors ([IntRep#3], [Gorodetski *et al*-02a]).

The MAS technology used in MAS DK comprises two main phases.

At the first one the application is specified in terms of a MAS DK specification language resulting in so called "*System Kernel*". If MAS has to be modified, it is necessary to modify *System Kernel* and to re-generate the respective software code.

At the next phase *System Kernel* is used for MAS deployment. At this phase the software of MAS is generated and software agents are situated in computers of the network according to the MAS specification in *System Kernel*.

A hierarchy of reusable software classes and generic data structures are comprised in the *Generic Agent* component. *Generic Agent* consists of the invariant scheme of agents' databases, the library of invariant Visual C++ and Java classes implementing basic mechanisms of data processing and the library of reusable Visual C++ and Java classes supporting message exchange. The latter includes implementation of KQML language, message content specification language based on XML, message templates, parsers needed to interpret message content.

The procedure of installation of the software agents in the target computers of the network is carried out in the semi-automatic mode. The only information that can be defined by the developer is to point out the name of hosts (*IP*-addresses), in which each agent must be situated as it is specified in *System Kernel*. Special software generates the parameters of the message templates to tune addressee(s) for each message, and copies the data associated with the agent to a folder in the host computer. Several simple operations follow up the last ones.

MAS DK consists of several specialized editors: Ontology editor, Editor of agent classes, Agent class ontology editor, Editor of agents' behavior scenarios, Editor of message templates, and Agents' cloning manager.

The agents generated by use of the developed technology have the same architectures. Differences are reflected in the content of particular agents' data and knowledge bases. Let us outline agents' architecture.

Each agent's neighborhood comprises other agents that it communicates with environment which it perceives, and, possibly, modifies, and user interacting with agents through his interface.

Receiver of input and *Sender of output messages* perform the respective functions. Messages received are recorded in *Input message buffer*.

The order of its processing is managed by *Input message processor*. In addition, this component performs syntax analysis and KQML messages interpretation and extracts the message contents.

The component *Database of agent's dialogs* stores for each input message its attributes that are identifiers, type of message and its source. If a message supposes to be replied it is mapped the respective output message when it is sent.

Meta-state machine manages the semantic processing of input messages directing it for processing by the respective *State machines*. One of the functionalities of this component is management of the parallel performance of agent's processes.

The basic computations of agent corresponding its role in MAS are executed by *State machine 1,…, State machine N*. Each of them is realizing a scenario of processing of some kind of input messages. *The set of these state machines realize attack scenario specified as a family of formal grammars.*

The selection of scenario and therefore the output result depend on the input message content and inner state of the *State Machine*. In turn, inner state of this *Machine* depends on pre-history of its performance; in particular, this prehistory is reflected in the state of agent's *Knowledge base* and *Database*. One more state machine called "*Self-activated behavior machine*" is changing its inner state depending on the state of the data and knowledge bases. In some combinations of these states it can activate functionality independently of input messages or state of the environment.

Each agent class is provided with a set of particular message templates according to its functionalities. These templates are formed through specialization of the of *Generic agent* component called *generic message template*. The developer carries out the specialization procedure with *Editor of message templates*, which, in turn, is a component of MAS DK. The message templates are specified in KQML language and specialization corresponds to the assignment to each template the respective performatives. At current phase of MAS DK development only standard KQML performatives are used. Communication component of each agent includes also data regarding potential addressees of messages of given template. The last data are assigned at the phase of agent class instances cloning. Message content is specified in XML. The XML content of each message is formed on-line according to the chosen scenario depending on the input message and agent's internal state. The package KQML+XML is configured by the procedure "Output message".

1.4.2. Object-oriented project of the Attack Simulator

The following main components of the Attack Simulator software compose the object-oriented project of the Attack Simulator [IntRep#3]:

(1) *The component setting the subject domain ontology* (*DomainOntology*). It serves for specifying and storing the notions, notion attributes, and values of notion attributes of the subject domain of computer network attack modeling. The subject domain ontology is filled in during the design-time stage by the MAS DK ontology editor. At this stage, notions (classes) of the ontology, notion (class) attributes, as well as meta-classes that unite notions into groups (they are not used in *DomainOntology*) are entered and modified through the user interface.

(2) *The component realizing the set of attack scenarios as a family of state machines* (*AttackModel*). Through specifying the so-called "applied" state machines, *AttackModel* determines the executable computer network attack scenarios. This component is used to specify the set of different classes of computer network attacks in the form of a family of state machines. States and transitions between states are the main elements of a state machine. The scheme of description of each state machine includes the following elements: name of state machine, its purpose and general description; identifier of the attacks ontology node to which the state machine corresponds; state machine diagram; main parameters of the state machine; parameters of transitions; transition conditions; and scripts.

(3) *The component interpreting the family of state machines* (*Engine*). It realizes the operation mechanisms of the "applied" state machines specified in the component *AttackModel*, and controls their behavior by an invariant meta- state machine. The applied state machines operates under the control of the meta- state machine. Meta- state machine creates examples of applied state machines, transforms state machines' scripts into the internal view, enables the communication between state machines' scripts and the component *DomainOntology*, executes the scripts, delegates control between applied state machines, and eliminates the examples of applied state machines after they complete their function.

(4) *The kernel component defining the attack task specification and supporting the interaction between main components of the prototype* (*AgentLib*). The component *AgentLib* is a sort of a "gateway" connecting the component *Engine* to the program components that realize specific applied tasks.

(5) *The user interface component for attack task specification* (*TargetObjectiv*). Attack specification includes intention, address of the attacked host (network), object of attack, and information about the object of attack (host or network) already known to the attacker.

(6) *The component calculating probability of the state machine transition to the next state* (*SMProb*). The component consists of three classes: (a) *Transition* responsible for each separate (unique) transition; (b) *Transitions* containing a probabilistic group that consists of instances of the class *Transition* that are responsible for each separate transition within this probabilistic group; and (c) *Prob_DB* responsible for working with the table *SrcProb*, the interaction with the classes of the probabilistic group and the classes of the separate transitions within the probabilistic group.

(7) *The user interface component for visualizing the process of attack realization* (*AtlogView*).

The component model of the software prototype of the Attack Simulator is represented in Fig.1.4.1.

The components *AgentLib*, *TargetObjectiv*, *SMProb* and *AtlogView* are the program implementing the prototype in VC++. The components *DomainOntology* and *AttackModel* (marked blue in Fig.1.4.1) are the semantic constituents of the prototype. They are described in terms of the structures represented by the MAS DK environment.

The diagram shows that the components *TargetObjectiv*, *SMProb* and *AtlogView* interact through calling the functions of the component *AgentLib*, which in turn is governed by *Engine*.

The object-oriented description of the software implementation of the state machine model and the interaction between the state machine model and the subject domain is represented (specified) in the component *Engine*.

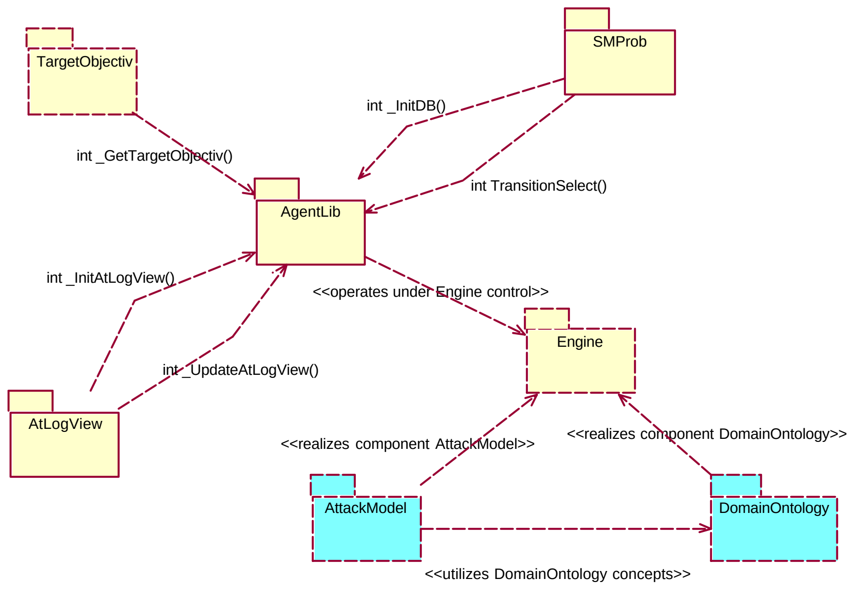


Fig.1.4.1. Component model of the software prototype of the Attack Simulator

1.5. Related works

The works relevant to attack modeling and simulation can be divided into the following groups:
(1) Works describing attacks and attack taxonomies.
(2) Works immediately coupled with network attack modeling and simulation.
(3) Works devoted to the description of attack languages.
(4) Works on evaluating intrusion detection systems (IDSs).
(5) Works on vulnerability assessment tools (scanners), signature and traffic generation tools.
But this list is not exhaustive. Let us consider the works relevant to attack modeling and simulation according to these groups.

1.5.1. Works describing attacks and attack taxonomies

Till now a lot of data about different security incidents is accumulated. There are a lot of publications in which attack cases are systematized in the form of *attack taxonomies*.

The following attack taxonomies can be distinguished:

- lists of attack terms ([Cohen-95], [Icove *et al*-95], [Cohen-97], [Howard-97], [Howard *et al*-98]);
- lists of attack categories ([Cheswick *et al*-94], [Ranum-97]);
- attack results categories ([Cohen-95], [Russell *et al*-91]);
- empirical lists of attack types ([Lackey-74], [Neumann *et al*-89], [Amoroso-94], [Lindqvist *et al*-97]);
- vulnerabilities matrices ([Amoroso-94], [Landwehr *et al*-94]);
- action-based taxonomies [Stallings-95];

- security flaws or vulnerabilities taxonomies ([Beizer-90], [Saltzer *et al*-75], [Hogan-88], [Aslam-95], [Dodson-96], [Krsul-98], [Power-96]);
- taxonomies of intrusions based on the signatures [Kumar-95]; incident taxonomies ([Howard-97], [Howard *et al*-98]).

The example of the incident taxonomy is a common language for security incidents suggested by Howard and Longstaff [Howard *et al*-98]. Three main high-level concepts of the language (an incident, an attack, and an event) are defined by seven groups of auxiliary low-level concepts (attackers, tools, vulnerabilities, actions, targets, unauthorized results, and objectives) (Fig.1.5.1). An attack is defined as "a series of intentional steps taken by an attacker to achieve an unauthorized result." An incident is defined as "a group of related attacks that can be distinguished from other attacks because of the distinctiveness of the attackers, attacks, objectives, sites, and timing."

Based on these taxonomies *we built our own taxonomy as an ontology comprising a hierarchy of intentions and actions of malefactors directed to implementation of attacks of various classes split into macro- and micro-levels.*

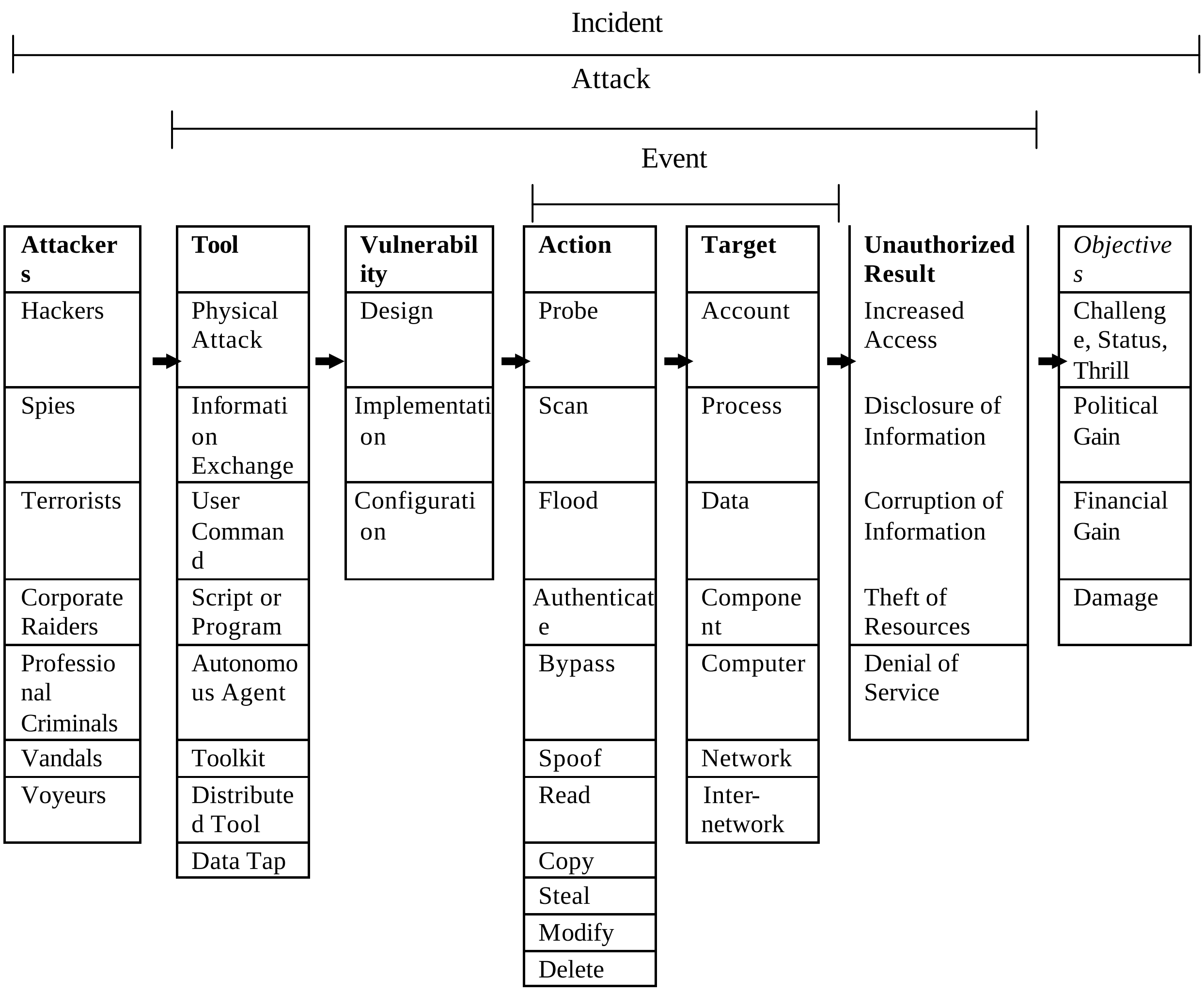


Fig.1.5.1. Computer and Network Incident Taxonomy [Howard *et al*-98]

1.5.2. Works immediately coupled with network attack modeling and simulation

In different works on *attack modeling and simulation*, as a rule, attack is considered as temporal orderings of actions ([Kumar *et al*-94], [Iglun *et al*-95], [Chung *et al*-95], [Kemmerer *et al*-98], [Amoroso-99], [Cohen-99], [Stewart-99], etc.).

In a temporal model of attack realization, an intruder begins with some initial action, and this action is followed by supporting actions, etc. [Amoroso-99]. Response and other actions may also be involved, and the security officer, normal users, other intruders, and so on may initiate these actions. The resultant sequence of actions models the exploitation of vulnerabilities to bring about the unauthorized security threat. This notion is represented in Fig.1.5.2 ([Amoroso-99]).

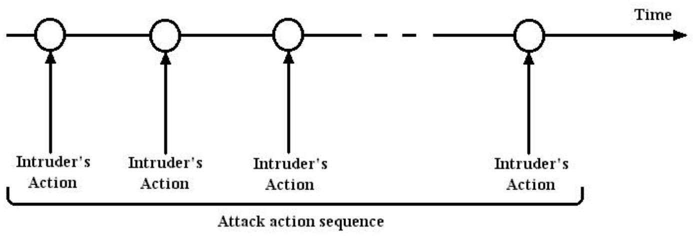


Fig.1.5.2. Temporal Model of Intrusion([Amoroso-99])

The model of intrusions based on *Colored Petri Nets* was proposed in [Kumar *et al*-94]. Each intrusion signature is expressed as a pattern that represents the relationship among events and their context. The notions of start and final states, and paths between them determine the set of event sequences. Intrusion patterns have preceding conditions and following actions associated with them.

In [Iglun *et al*-95] the *state transition analysis technique* was developed to model host-based intrusions. This paper describes computer penetrations as sequences of actions that an attacker performs to compromise the security of a computer system. Attacks are described by using state transition diagrams. A description of an attack has a "safe" starting state, zero or more intermediate states, and (at least) one "compromised" ending state. States are characterized by means of assertions describing aspects of the security state (file ownership, user identification, user authorization).

An approach to *simulate intrusions in sequential and parallelized forms* was suggested in [Chung *et al*-95]. This paper presents an algorithm for transforming a sequential intrusive script into a set of parallel scripts, which simulate an intrusion.

The paper [Kemmerer *et al*-98] describing IDS NetSTAT presents *formal models of both network and attacks*. These models permit to determine "which network events have to be monitored and where they can be monitored". The NetSTAT approach extends the state transition analysis technique [Iglun *et al*-95] to network-based intrusion detection in order to represent attack scenarios in networks.

In [Cohen-99] a simple network security *model "Cause-Effect Model of Information System Attacks and Defenses"* was suggested. It is composed of network model represented by node and link, cause-effect model, characteristic functions, and pseudo-random number generator (Fig.1.5.3). However, cyber attack and defense representation which is based on cause-effect model [Cohen-99] is very simple.

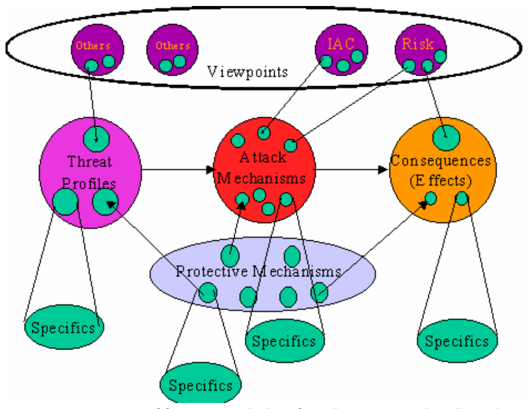


Fig.1.5.3. Cause-effect model of cyber attack ([Cohen-99])

In [Yuill *et al*-99] and [Yuill *et al*-00] the *descriptive models of the network and the attacker*'s capabilities, intentions, and courses-of-action are described. These models are used to identify the devices most likely to be compromised. Principles from economics are used to predict the attacker's behavior.

One of the *conceptual models of computer penetration* was presented in [Stewart-99]. The paper compares the traditional and new attack paradigms. Traditional attack paradigm includes phases of "information gathering", "exploitation", and "metastasis". The metastasis phase of the attack, as defined by Cheswick and Bellovin [Cheswick *et al*-94], can be logically separated into sub-phases of "consolidation" and "continuation". During consolidation the attacker must remove evidence of the entry onto the host, escalate their privilege to the highest level, and enable remote unauthorized access. On continuation sub-phase he tries to deep the penetration to other hosts.

The core of the distributed metastasis methodology suggested in [Stewart-99] is a desire to utilize the distributed nature of network environment, and to perform an automation of the metastasis phase of the traditional attack process. A distributed agent-based approach can be applied to the metastasis phase of the traditional attack methodology to reap appreciable benefits for an attacker. Multiple agents distribute on network, reside on topologically disparate hosts, communicate attack-relevant information, and penetrate the network.

We used in our formal model the temporal orderings of actions and proposed multi-agent based approach for modeling of attacks.

In ([Huang *et al*-98], [Schneier-99] [Moore *et al*-01], [Dawkins *et al*-02]) attacks are described and modeled in a structured and reusable *"tree"-based form*.

In [Huang *et al*-98] a *high-level conceptual model of attack based on the intruder's intent* (attack strategy) is presented. The large-scale distributed intrusion detection is compared with the task of battleground management. The paper determines intrusion intention as the goal-tree. The ultimate goal of intrusion corresponds to the root node. Lower level nodes represent alternatives or ordered sub-goals in achieving the upper node/goal. The end nodes (leaves) are sub-goals. They can be substantiated by events generated in different environments. The "OR", "AND", and "Ordered-AND" constructs are used for representation of temporal sequences of intrusion intentions. For example, Fig.1.5.4 shows a possible representation of flooding/spoofing/sniffing sequences [Huang *et al*-98].

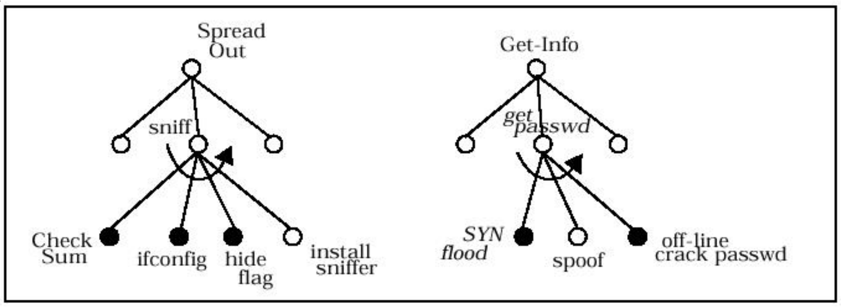


Fig.1.5.4. Flooding/spoofing/sniffing and possible directions of intrusion development

The paper [Schneier-99] also suggests a formal approach for describing computer attacks. This approach is called "*Attack trees*". "*AND*" and "*OR*" nodes are used in attack trees. *OR* nodes are alternatives. *AND* nodes represent different steps toward achieving the same goal.

A more comprehensive work using so-called *"tree"-based approach* is proposed in [Moore *et al*-01]. This paper describes a means for documenting attacks in a form of attack trees. A node of an attack tree is decomposed either as an *AND*-decomposition (a set of attack sub-goals, all of which must be achieved for the attack to succeed), or as an *OR*-decomposition (a set of attack sub-goals, any one of which must be achieved for the attack to succeed). "An enterprise typically has a set, or forest, of attack trees that are relevant to its operation. The root of each tree in a forest represents an event that could significantly harm the enterprise's mission. Each attack tree enumerates and elaborates the ways that an attacker could cause the event to occur" [Moore *et al*-01]. Two structures are used for attack representation: (1) *attack pattern* (characterizing an individual type of attack), and (2) *attack profile* (organizing attack patterns to make it easier to apply them). Each attack pattern contains: the overall goal of the attack, a list of preconditions for its use, the steps for carrying out the attack, a list of post-conditions that are true if the attack is successful. Attack profiles contain a common reference model, a set of variants, a set of attack patterns, and a glossary of defined terms and phrases.

As in [Huang *et al*-98] and [Moore *et al*-01] *we apply intension- and tree-based attack strategy representation, but "go further" using for node decomposition a formal framework based on context-free stochastic attribute grammars implemented in terms of state machines.*

A *model to evaluate survivability of networked systems* after network incidents was developed in [Moitra *et al*-01]. The model consists of three sub-models. The first one simulates the occurrence of attacks or incidents. The second one evaluates the impact of an attack on the system depending on the attack type and the protection system maturity. The third one assesses the survivability of the system. The model of incidents is determined as a marked, stochastic process, where the incidents are the events that occur at random points in time, and the event type is the mark associated with an incident. Each occurrence time t_k of the k-th incident in a temporal point-process has a mark j_k associated with it, where j_k will have values in a specified space. The mark has to take into account the severity of the incident and the possibility of single, or multiple and simultaneous attacks. Therefore the mark space will be 2-dimensional, characterized by type (severity) and number-of-attackers.

Besides attack generation model, our approach includes also the model of attacked computer network that evaluates the impact of an attack on the network hosts and generates reaction of the network. The attacked network is considered as environment that reacts on the malefactors' actions. The variance of attacks is ensured by the random choice of the grammar productions (or, what is the same, the state machine transition rules). The peculiarity of any attack is that the malefactor's strategy depends on the results of the intermediate actions.

The paper [Chi *et al*-01] describes the cyber attack modeling and simulation methodology based on *SES/MB* (system entity structure and model base) framework, *Discrete Event Simulation* (DEVS) formalism [Zeigler 90], and experimental frame concept underlying the object-oriented software environment. This simulation methodology allows classifying threats, specifying attack mechanisms, verifying protection mechanisms, and evaluating consequences.

The paper [Templeton et al-00] describes a flexible extensible *model for computer attacks*, a language for specifying the model, and how it can be used in security applications such as vulnerability analysis, intrusion detection and attack generation.

In [Goldman-02] the task of *modeling and simulation of intelligent, reactive attackers* is described. The suggested computer network attack model uses an action representation based on the Golog *situation calculus* [Levesque *et al*-97] and *goal-directed* procedure invocation. Using the situation calculus, the developed attack simulator can project the results actions with complex preconditions and context-dependent effects. The goal-directed invocation permits to express attacker plans like "first attain root privilege on a host trusted by the target, and then exploit the trust relationship to escalate privilege on the target". Goldman has designed components of a stochastic attack simulator which can simulate some goal-directed attacks on a network.

Several works propose and use a concept of *attack graphs* ([Dacier-94], [Phillips *et al*-98], [Ritchey *et al*-00], [Swiler *et al*-01], [Jha *et al*-01], [Sheyner *et al*-02a], [Jha *et al*-02], [Sheyner *et al*-02b], etc.).

Dacier [Dac94] proposed the concept of privilege graphs, where each node represents a set of privileges owned by the user and arcs represent vulnerabilities. Privilege graphs are explored to construct attack state graphs, which represent different ways in which an intruder can reach a certain goal, such as root access on a host. An experimental evaluation of this framework is described in [Ortalo *et al*-01].

Phillips and Swiler built a tool for generating attack graphs by forward exploration starting from the initial state of the attack ([Phillips *et al*-98], [Swiler *et al*-01]) and used it for attack network vulnerability assessment.

Ritchey and Amman [Ritchey *et al*-00] also applied model checking for vulnerability analysis of networks. They used the unmodified model checker SMV [SMV].

In ([Sheyner *et al*-02a], [Jha *et al*-02], [Sheyner *et al*-02b], [Jha *et al*-01]) an *automated technique for generating and analyzing attack graphs* is represented. The technique is based on symbolic model checking algorithms ([Clarke *et al*-00], [SMV], [NuSMV]), letting construct attack graphs automatically and efficiently. Since a generic state machine model is used, the authors can model not just attacks, but also seemingly benign system events (e.g., link failures and user errors) and even system administrator recovery actions. The authors implemented the technique in a tool suite and tested it on a small network example, which includes models of a firewall and an intrusion detection system. They use a symbolic model checker (i.e., NuSMV [NuSMV]) that works backward from the goal state to construct the attack graph. Authors suggested applying this technique and the tool suite for vulnerability analysis of a network. A typical process for vulnerability analysis proceeds as follows. First, vulnerabilities of individual hosts (using scanning tools, such as COPS and Nessus Scanner) are determined. Using this local vulnerability information along with other information about the network, such as connectivity between hosts, they then produce attack graphs. Each path in an attack graph is a series of exploits, which they call atomic attacks, that leads to an undesirable state, e.g., a state where an intruder has obtained administrative access to a critical host. Then further analyses (such as risk analysis, reliability analysis, or shortest path analysis) on the attack graph to assess the overall vulnerability of the network are performed.

As in these works our approach can be also used to build different attack graphs, but it uses stochastic formal-grammar-based specification of the malefactor's intentions and scenarios of network attacks.

1.5.3. Works devoted to the description of attack languages

Attack languages are used with the purpose of attack recognition, analysis of the relations between various attacks, response on them and documenting of intrusions. Besides, attack languages can be used for fixing the scenarios and prehistory of attacks, and also for reproduction of attacks with the purposes of testing intrusion detection systems ([Vigna *et al*-00], [Eckmann *et al*-00]).

Attack languages are classified using various tags. In particular, in [Vigna *et al*-00] the classification of the attack description languages is offered, according to which six types of languages are entered:

- event languages;
- exploit languages;
- reporting languages;
- detection languages;
- correlation languages;
- response languages.

Event languages ([BSM-91], [Jacobson et al-00], [Bishop-95], etc.) describe the format of events used during the detection process.

Exploit languages ([CASL-98], [Deraison-99], etc.) are used to describe the stages to be followed to perform an intrusion.

Reporting languages ([Feiertag et al-99], [Curry-00]) describe the format of alerts produced by the IDS.

Detection languages ([Kumar *et al*-95], [Paxson-98], [Roesch-99], [Turner et al-00], [Eckmann *et al*-00], [Me-98]) allow the expression of the manifestation of attacks.

Correlation languages permit analysis of alerts provided by several IDS.

Response languages are used to express countermeasures to attacks.

Examples of specific attack languages are represented in Table 1.5.1 [Michel *et al*-01].

Let us consider, for example, how attacks can be represented in two of the advanced attack languages: STATL [Eckmann *et al*-00] and AdeLe [Michel et al-01].

STATL is an extensible attack language designed to support intrusion detection [Eckmann *et al*-00]. The STATL provides constructs to represent an attack as a composition of states and transitions. States are used to characterize different snapshots of a system during the evolution of an attack. A transition has an associated action that is a specification of the event that may cause the scenario to move to a new state. The space of possible relevant actions is constrained by a transition assertion, which is a filter condition on events that may possibly match the action.

AdeLe is designed to model a database of known attack scenarios [Michel *et al*-01]. An ADeLe description looks like a function in *C* programming language with name and parameters. The description body is made up of three parts: exploit part, detection part, and response part.

Table 1.5.1. Examples of specific attack languages

	Event languages	Exploit languages	Reporting languages	Detection languages	Correlation languages	Response languages
BSM [BSM-91]	+					
Tcpdump [Jacobson *et al*-00]	+					
Bishop [Bishop-95]	+					
CASL [CASL-98]		+				
NASL [Deraison-99]		+				
CISL [Feiertag *et al*-99]			+			
IDMEF [Curry-00]			+			
Kumar [Kumar *et al*-95]				+		
BRO [Paxson-98]				+		
Snort [Roesch-99]				+		
SNP-L [Turner *et al*-00]				+		
STATL [Eckmann *et al*-00]				+		
GasSATA [Me-98]				+		
LAMBDA [Cuppens *et al*-00]				+	+	
AdeLe [Michel *et al*-01]				+	+	+

The exploit part represents the attacker's point of view. It is composed of three sections: pre-condition, attack, and post-condition. The pre-condition section expresses the requirements for launching the attack. These are data about the target operating system, installed software, the vulnerabilities, the level of privilege needed by the attacker to launch a successful attack, etc. The attack section determines the source code of the attack that can be expressed in different languages (*"C"*, *"C++"*, *"Perl"*, *"Casl"*, *"Nasl"*, etc.). In order to represent the variants of the same attack, operators *Non_ordered*, *One_among*, and *Subset_of* are introduced. In the post-condition section, what the attacker has obtained is expressed (an increased level of privilege, disclosure of information, corruption of information, denial of service, etc.).

Our formal language concerns mostly to the exploit and event languages, because it is used to describe attack stages and the format of events generated. Our attack representation language includes parts used for description of attack pre-conditions, attack intentions and actions, formats of actions of terminal level, and post-conditions (states of the attacked hosts).

1.5.4. Works on evaluating intrusion detection systems

Several publications devoted to *evaluation of IDSs* have considered attack modeling and simulation issues.

The papers ([Puketza *et al*-96], [Puketza *et al*-97]) describe a *methodology and software tools for testing IDSs*. The methodology consists of using scripts to generate both background traffic and intrusions with multiple streams of activities.

In *evaluations of IDSs* described in ([Lippmann *et al*-98], [Lippmann *et al 1*-00], [Lippmann *et al 2*-00], [Kendall-99], [Korba-00], [Das-00]), investigators were given sensor data in the form of sniffed network traffic, audit data, and file-system snapshots. The test network consisted of real and simulated machines. The attack set includes *denial-of-service* attacks, *remote-to-local* attacks, *user-to-root* attacks, *probe attacks, insider* attacks, *console-based* attacks, *a man-in-the-middle* attack, and an attack using *macro code*.

The report [Debar *et al*-98] discusses issues associated with *generation of suitable background traffic*, noting the difficulties associated with alternatives, including developing accurate models of user and server behavior, using test suites designed by operating system developers to exercise server behavior and using recorded “live” data. Attacks are obtained from an internally maintained vulnerability database.

The Air Force Rome Lab has built *a real-time test bed* [Durst *et al*-00].

Like systems in general, IDSs can be evaluated in various ways, such as *benchmarking* or *modeling*. The papers ([Alessandri *et al*-01], [Durst *et al*-00], [Lippmann *et al 1*-00], [Lippmann *et al 2*-00]) point out that benchmarking real IDSs is not generic and systematic enough for evaluation needs. John McHugh has criticized the benchmarking approach according to exactly the same reasons [McHugh-00]. Because of these insufficiencies in [Alessandri *et al*-01] another approach is investigated. It consists of *comparing and evaluating IDSs at the level of their specification* rather than at the level of their implementation. This approach describes IDSs by formalizing their characteristics, and does not attempt to describe the implementation of the intrusion detection algorithms used.

Our approach also presumes that IDSs can be evaluated and verified at different phases of their development and implementation. The more detailed level of attack representation is used in the attack model the more advanced level of IDS is evaluated.

1.5.5. Works on vulnerability assessment tools (scanners), signature and traffic generation tools

Now a lot of *tools can automate the vulnerability discovery process*. For example, Internet Scanner (ISS), NetRecon (Axent Technologies), CyberCop Scanner (Network Associates), Nessus Security Scanner, etc.

Also there are a lot of *signature and traffic generation tools*: WebRamp, MS WCAT, Hailstorm, IDS Informer, nidsbench, SmartBits, FlameThrower, Stick, Fragrouter, etc. But the majority of these tools are doing only simulated pseudorandom malicious packets. As Marcus Ranum marked by discussions on focus-ids@securityfocus.com: “Make sure that you're not only generating “signatures” but that they are within the context of apparently valid sessions - otherwise you're actually benchmarking an IDS' ability to detect false positives, not real attacks.”

According to our sight, Hailstorm (Cenzic) and IDS Informer (BLADE Software) have most interesting properties.

Hailstorm [Hailstorm-00] is a traffic generation product aimed at simulating severe-load conditions, malicious network attacks and robust integrity-analysis of online applications. Hailstorm generates traffic based on patterns specifying how a packet is to be generated over the network.

IDS Informer [IDS Informer-01] has been designed to allow launch S.A.F.E. (Simulated Attacks For Evaluation) attacks, and to report on how the IDS system coped and responded to those attacks. The S.A.F.E. process builds the attacks based on previously recorded real attacks.

In our approach we tried to realize the malicious and background traffic on the terminal levels of the attack model within the context of valid sessions.

As one can see from the review of relevant works, the field of attack modeling and simulation has been delivering significant research results to date, nevertheless the publications reflect a beginning phase of research. Perhaps this is due to the extreme complexity of the network attack and computer networks. “There is no widely accepted information physics that would allow making an accurate model, and the sizes of the things we are modeling are so large and complex that we cannot describe them with any reasonable degree of accuracy” [Chi *et al*-01]. The high cost of running real-world attacks, the limited extent to which they exercise the space of actual attacks, and the high potential for harm from a successful attack conspire to make some other means of analysis an imperative [Cohen-99]. *We have developed a strict formal model and techniques for attack modeling based on stochastic formal-grammar-based specification of the malefactor’s intentions and scenarios of network attacks on the macro and micro levels.* Our approach has applied the results of reviewed relevant works, but is evolving own theoretical and practical ideas about *stochastic formal grammar and multi-agent based* attack modeling and simulation.

1.6. Conclusion

The Chapter presents summary of the theoretical results received in the research according to the Task 1 of Project 1994P. Let us summarize the main results.

1. *Main concepts of standard remote attacks on computer networks, review, classification and analysis of the computer network attacks* were presented.

- The existing *attack taxonomies* analyzed are multifold, but can not ensure basis for the project goals achievement. The incident taxonomies are the most prominent for our goals, but they require a more elaboration with emphasis on remote actions.
- The to date suggested *classifications of the standard remote attacks* allow to classify attacks by character of effect, purpose of effect, condition of beginning of the effect realization, presence of feedback with the attacked object, layout of the subject of attack concerning the attacked object, layer of standard ISO/OSI model, on which the effect is carried out, the object, on which is carried out an effect, attack complexity level.
- Eight *typical classes of the remote attacks* are marked out in this research: (1) analysis of the network traffic (or listening of a data link by means of special tools – sniffers), (2) network scanning (probing), (3) substitution of the trusted object of the network and transmission of the messages "on its behalf" with appropriation of trusted object access rights, (4) implantation of the false object in a network, (5) denial of service, (6) unauthorized access from a remote machine by guessing password, (7) unauthorized access to local super user (root) privileges, (8) remote initiation of applications.
- To our opinion, the research results make it deeper the understanding of the environment within which a modern computer network is operating, potential inside and outside attacks, degrees of the particular attack danger with respect to the possible aftereffects, and degree of the network vulnerability. The proposed attack classification constitutes the starting points for development of the formal approach and the tool for simulation of attacks against computer networks. The presented results enable the development of the conceptual descriptions of representative set of network attacks on a macro-level and to develop their formal models.

2. *Scenario-based specifications of computer network attacks* were introduced. This result corresponds to the phase of a complex system design on which the conceptual model of the research area is specified. Particularly,

- An *attack model* is understood as a formal object having a likeness in basic properties with regard to real-life attacks, serving for investigations by means of fixing known and obtaining new information about attacks. A *formal model of attacks* is a collection of mathematical dependences specifying a broad spectrum of attacks and allowing study of them formally and via simulation.
- The developed *conceptual model of computer network attacks* includes two levels: (1) macro-level and (2) micro-level. On the macro-level, a common attack scenario is defined. It includes partially ordered set of malefactors' intentions and actions. The micro-level specifies more detailed representation of attack as a sequence of low level commands, system calls, etc. Each step of the macro-level scenario is represented as a sequence of simple operations (events) on micro-level.
- Each sequence of attack steps can be considered as a "word" belonging to a formal language that, in turn, can be specified in terms of a *formal grammar*. The grammar may be regenerated by use of formal methods inductively on the basis of sample of cases, and than it can serve as a formal model of attacks on the macro level.
- The *scenario-based models of eight network attack classes* are determined. These classes are the followings: (1) analysis of the network traffic, (2) network scanning (probing), (3) substitution of the trusted object of the network and transmission of the messages from its name with appropriation of its access rights, (4) implantation of the false object in a network, (5) denial of service, (6) unauthorized access from a remote machine by guessing password, (7) unauthorized access to local super user (root) privileges, and (8) remote initiation of applications. Each scenario is described by a set of admissible sequences of steps determining an attack class on both macro and micro levels.

3. *Techniques for case-based regenerating of the formal grammar specifying models of the attacks* were defined. Particularly,

- It is shown that the scenario of a computer network attack can be specified formally as a certain *formal grammar*. That grammar can be used both as the model generating the examples of attacks and as the model for recognition of attacks based on syntactic analysis. For practical implementation of attacks modeling systems, it is necessary to construct such grammars based on sample of cases of the already implemented attacks.
- Formally, the *task of synthesizing a grammar* consists in creating the algorithm to recover its production rules based on the finite set of words of the language (represented by the grammar) that specifies attack cases, and possibly, based on the finite set of words from the supplement to this language.
- The grammar that generates scenarios for computer network attacks can be synthesized: (1) through *inductive recovery* based on the set of cases through the use of formal methods; (2) by an *expert* who possesses knowledge of the malicious party's intentions and the possible ways these intentions can be realized; (3) through *combining* the two above methods.
- Two groups of *algorithms* can be used *for grammar recovery*: (1) enumeration grammar recovery algorithms; (2) induction grammar recovery algorithms. The inductive grammar recovery methods are deemed the most adequate for the purposes of recovering grammars that specify computer network attacks; specifically, the inductive method for recovering regular grammars on the basis of positive examples (the *Feldman method*). This method consists in constructing a non-recursive grammar that creates precisely those strings that were present in the training example, and then arriving at a simpler recursive grammar that generates all the strings of the positive examples and an infinite amount of other strings.
- The analyzed examples of computer network attacks have shown *practical applicability of the grammar recovery algorithms* suggested. It was shown that the synthesized grammars can generate attacks corresponding to the training data used for development of the grammar. But the grammars developed through combining different productions are capable of generating the attacks that are not included in the training cases. This can expand capabilities of the intrusion detection system and (or) attack simulator based on these grammars.

4. *Mathematical methods and techniques realizing attack modeling and simulation* were developed. The conceptual justification of the chosen approach, problem domain ontology, specification of the basic components composing attack model, their interaction in simulation procedure, and examples of the network attacks specifications have been elaborated.

- The conceptual exploration of the computer network attacks has discovered the following *peculiarities of planning and execution of attacks*, influencing on choice of a formal model of attacks: (1) any attack is directed to the concrete object and, as a rule, has a quite definite intention; (2) an attack intention can be represented in terms of partially ordered set of lower-level intentions and actions realized in multiple ways; (3) an attack development greatly depends on the response of the attacked computer network, choice of attack continuation is almost always non-deterministic; (4) an attack development scenario cannot be definitely specified beforehand, since any attack depends on many uncertainties. T*he basic elements of the developed approach to network attacks modeling and simulation are* the attack goals; the ontology of the computer network attacks domain; the family of enclosed attributed stochastic context-free grammars describing different attack scenarios on the basis of the ontology of the computer network attacks domain; the family of state machines for the computational interpretation of the grammars; the model of the attacked computer network including a generalized model of the computer network structure, host models, a model for calculating the probability of successful actions (attacks) against the host and a host response model.
- *The developed approach to the modeling and simulation of network attacks* consists in the following: in accordance with the attack goal a sequence of actions imitating attacks is generated on the basis of a family of enclosed attributed stochastic context-free grammars (or a family of state machines based on them) in accordance with the model of the attacked computer network. The variance of attacks is ensured by the random choice of the grammar productions (or the state machine transition rules) and randomization of the success of separate actions.

5. *Object-oriented project of the Attack Simulator* was developed.

- For development of the *object-oriented project*, the technology and software tool called *Multi-agent System Development Kit* (MAS DK) was used. This tool is implemented on the basis of Visual C++ 6.0, JAVA1.3 and XML programming languages. The MAS technology comprises two main phases: (1) Specification of the application in terms of a MAS DK specification language resulting in so-called "*System Kernel*" containing the complete information about the developed application; (2) Usage of *System Kernel* for MAS deployment (at this phase the software of MAS is generated and software agents are situated in computers of the network according to the MAS specification in *System Kernel*). The agents, generated by the usage of the developed technology, have the same architecture. Differences are reflected in the content of particular agents' data and knowledge bases.
- The suggested *approach for the object-oriented design of the Attack Simulator* is based on the usage of the following main components: (1) *DomainOntology* setting the subject domain ontology; (2) *AttackModel* realizing of the set of attack scenarios as a family of state machines; (3) *Engine* fulfilling interpretation of the state machines; (4) *AgentLib* defining the attack task specification and supporting the interaction with main components of the prototype; (5) *TargetObjectiv* for attack task specification; (6) *SMProb* for calculating probability of the state machine transitions; (7) the user interface component for visualizing the process of attack realization (*AtlogView*).

6. *Works relevant to attack modeling and simulation* was analyzed and proposed and developed approach was compared with them.

- The relevant works can be divided into five groups: (1) works describing attacks and attack taxonomies; (2) works immediately coupled with network attack modeling and simulation; (3) works devoted to the description of attack languages; (4) works on evaluating intrusion detection systems; (5) works on vulnerability assessment tools (scanners), signature and traffic generation tools.
- One can see from the review of relevant works that the field of attack modeling and simulation has been delivering significant research results to date, nevertheless the publications reflect a beginning phase of research. Perhaps this is due to the extreme complexity of the network attack and computer networks.
- In our approach we have applied the results of reviewed relevant works, but have developed own specific theoretical and practical ideas about *stochastic formal grammar and multi-agent based* attack modeling and simulation.

Chapter 2. Software prototype of the Attack Simulator implementing theoretical results of the research and its evaluation

Abstract. The Chapter overviews the software prototype of the Attack Simulator developed and its evaluation results. The generalized architecture of Attack Simulator prototype is described. The main components of the program prototype, including Agent Hacker and Network Agent, as well as their functional capabilities and specific features of implementation are discussed. State-machine based descriptions of main components of Attack Simulator are determined. A detailed specification of all notions of the application domain ontology is fulfilled. The main components of generic Hacker Agent are outlined. These components include fragment of general ontology of computer network attack generation application domain, that is used by Hacker Agent in its operation, state machine model that realizes Hacker Agent's behavior scenarios, a set of scripts directing the Hacker Agent's behavior, program modules that realize the necessary functions and user interfaces (attack goal specification, calculation of the probabilistic hacker's behavior, network traffic generation, visualization of the attack scenario development). The main components of the generic Network Agent are sketched. They comprise a fragment of the general computer network attack application domain ontology that is used by the Network Agent, component of specification of computer network configuration, state machines model that realizes Network Agent's behavior scenarios, scripts for controlling the Network Agent in the process of forming a response to Hacker Agent's attack actions. The results of experiments that demonstrate the Attack Simulator efficiency in generating various attacks scenarios against computer networks with different configuration and security policy used are described in detail. The attack generation processes on macro (generation malicious actions against computer network model) and micro levels (generation malicious network traffic against real computer network) are analyzed.

2.1. Generalized architecture of Attack Simulator prototype

The software prototype for computer network attack simulation is built as a multi-agent system that uses two classes of agents. The agent of the first class simulates defense system of the attacked computer network ("Network Agent") and the second one simulates a hacker performing attack against computer network ("Hacker Agent"). In the developed prototype each agent class has single instance although the developed technology makes it possible to simulate a team of hackers and a team of agents responsible for computer network security.

The aforementioned agents are implemented on the basis of the technology supported by Multi-Agent System Development Kit (MASDK) that is a multi-agent software tool aiming at support of the design and implementation of multi-agent systems of a broad range [Gorodetski *et al*-02a]. The developed and implemented simulator comprises the multitude of reusable components generated by use of the MASDK standard functionalities and application-oriented software components developed manually in terms of programming language MS Visual C++ 6.0 SP 5. Fig.2.1.1 illustrates the general prototype architecture.

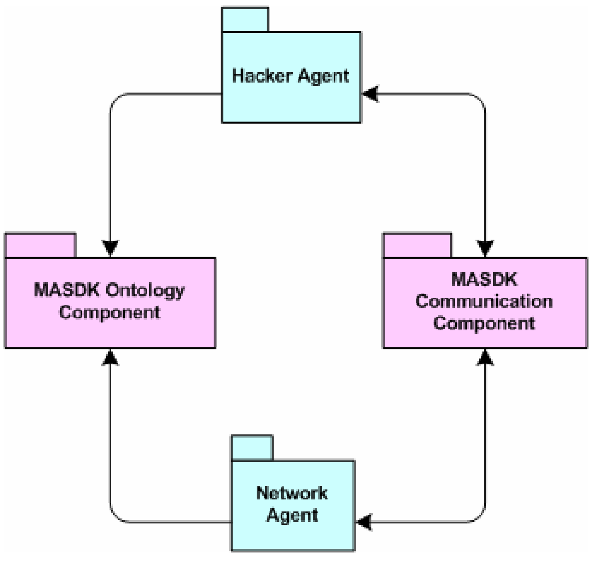


Fig.2.1.1. General Architecture of Attack Simulator

Each agent operates using the respective fragment of the application ontology that is designed by use of an editor of MASDK facilities. The interaction between agents in the process of attack simulation is supported by the communication environment, which design and implementation is also supported by MASDK.

It is noteworthy to note that the first version of the prototype (see Interim Report 3 [IntRep#3]) was implemented as a system that consisted of a single agent simulating a hacker's activity whereas computer network security system was simulated as a reactive system. In such architecture there were no need to situate both attacking Hacker Agent and computer network security reactive system in a communication environment. In the version of the prototype presented in this Report, the communication component plays a very important role. Indeed, the knowledge bases of Network Agent and Hacker Agent are implemented as two separate entities. An advantage of such a knowledge representation makes it actually possible to simulate adversary interactions. Such a model adequately implements interactions of the both above opposite sides. In it, while simulating an attack in order either to obtain response providing it with the needed information (on the reconnaissance stage) or to perform an attack action (on the threat realization stage) Hacker Agent sends a certain message to the Network Agent. The Network Agent, like this takes place in real-life interactions, analyzes the received message and forms a responsive message. This message is formed based on the Network Agent's knowledge base that models the network configuration and all its attributes needed to simulate the real-life response. The Network Agent's knowledge base also uses information about possible existing attacks and reaction of the network on them.

The key components of both agents correspond to so-called kernels that are the modules written in C++ and compiled into a dll. These components provide interface between the part of the software written in C++ and the components implemented through use of MASDK. The kernels provide interfaces to the respective fragments of the application ontology, and initialize the state machines, which in turn execute their scripts.

Let us consider *the main components of the Hacker Agent*.

The component model of the Hacker Agent is shown in Fig.2.1.2. It comprises the following main components:

(1) the core (Agent hacker Kernel);

(2) fragment of the application domain ontology;

(3) state machines model;

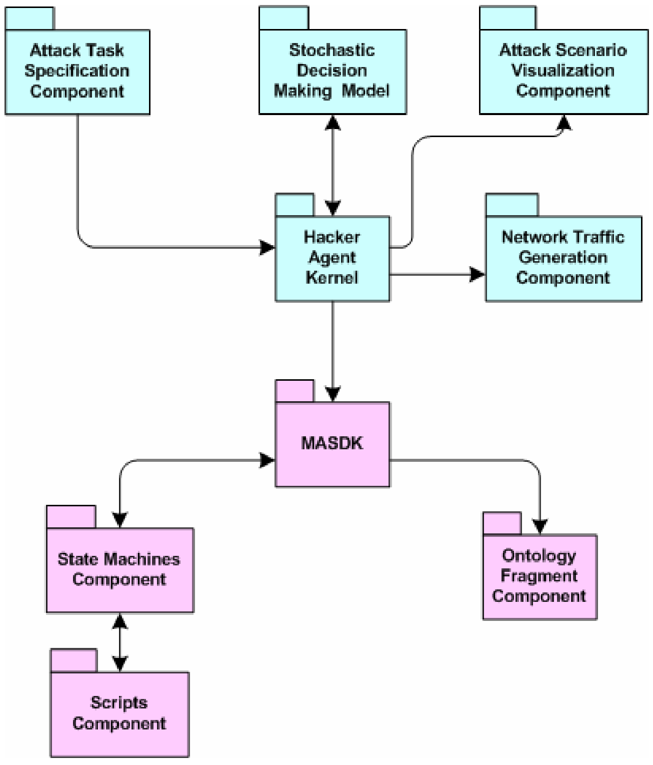
(4) scripts;

(5) attack task specification component;

(6) probabilistic (stochastic) decision-making model with regard to the further actions;

(7) network traffic generator;

(8) visualization component of the attack scenario development.


Attack Task Specification Component
Stochastic Decision Making Model
Attack Scenario Visualization Component
Hacker Agent Kernel
Network Traffic Generation Component
MASDK
State Machines Component
Ontology Fragment Component
Scripts Component
Fig.2.1.2. Component model of the Hacker Agent

The kernel of the Hacker Agent (Hacker.dll) contains a standard set of functions needed for exploiting ontology and running state machines. It is also provided with functions that call specification of attack task, compute next state-machine transition as well as initiate and perform visualization of the attack development.

Fragment of the application domain ontology specifies a set of notions and attributes used by the Hacker Agent. The application domain ontology is considered in section 2.3, and its fragment used by the Hacker Agent, is described in paragraph 2.4.1.

State machines model component is used for specification of the Hacker Agent behavior including decision making mechanism used by the Hacker Agent's to choice the next steps of action. The state machine model of the Hacker Agent is built on the basis of attribute stochastic grammars, and consists of over 50 nested state machines (the state machines model of the Hacker Agent is described in paragraph 2.4.2).

Script component specifies the set of scripts that can be performed by the Hacker Agent's state machines. The description of the scripts used by the Hacker Agent is given in paragraph 2.4.2 and Appendix 1.

Attack task specification component provides user with interface needed for him to specify attack attributes. This component is described in paragraph 2.4.3.

Probabilistic decision-making model is used to determine the Hacker Agent's further actions in attack generation. It is considered in paragraph 2.4.4.

Network traffic generator is used to form the flow of network packets for several classes of attacks directed to the hosts according to the attack specification. This component is initiated through calling the particular kernel function of the scripts of those states, for which the network traffic has to be generated. The network traffic generation component is described in detail in paragraph 2.4.5.

Visualization component of the attack scenario development is used for visual representation of the attack progress, corresponding to each action of attacker and respective response of the Network Agent. The response may be effective (the attack action was successful in part or in full), or ineffective (no response message, or the message saying that the attack was blocked by the firewall). This component is described in more detail in paragraph 2.4.6.

Let us consider the *main components of the Network Agent*. It is depicted in Fig.2.1.3. They are as follows:

1) the core (Network Agent Kernel);
2) fragment of the application domain ontology;
3) state machines model component;
4) scripts component;
5) network configuration specification component;
6) firewall model (implementation) component;
7) generator of the network's response to attack action.

Network Agent Kernel (NetAgent.dll) contains the standard set of functions for processing the application domain ontology and the state machine model, as well as the functions used for specification of network configuration through user interface, for the firewall model initialization, and for computation of the network's response to an attacking action.

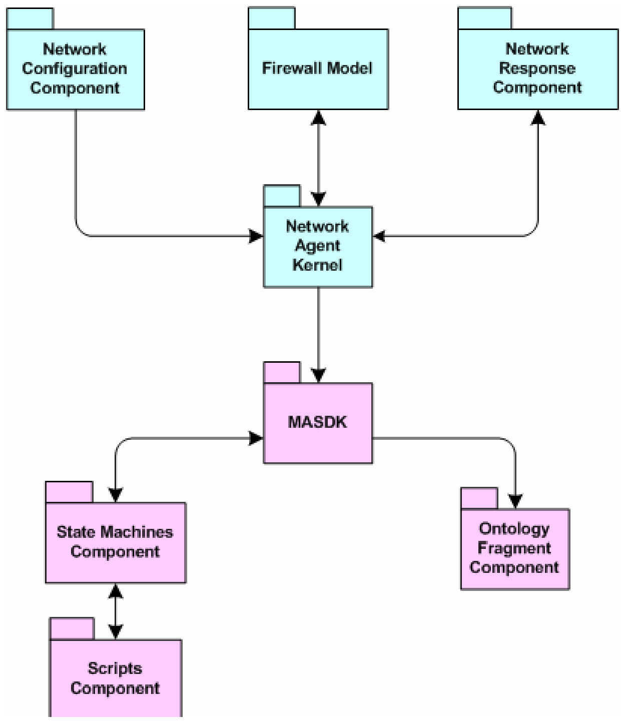

Network Configuration Component
Firewall Model
Network Response Component
Network Agent Kernel
MASDK
State Machines Component
Ontology Fragment Component
Scripts Component

Fig.2.1.3. Component model of the network agent

Fragment of the application domain ontology determines a set of notions and attributes used by the Network Agent. The application domain ontology is reviewed in detail in section 2.3, and its fragment used by the Network Agent is described in paragraph 2.5.1.

State machines model of the Network Agent's specifies its behavior. This state machines model mostly performs communication functionality. This component specifies the actions corresponding to the incoming message receiving, their classification, processing, and sending the response. The state machine model of the Network Agent is described in paragraph 2.5.3.

Scripts component specifies a set of scripts initialized from the state machine model of the Network Agent. The description of scripts used by the Network Agent is given in paragraph 2.5.3 and Appendix 2.

Network configuration specification component is used for the specification of a set of user interfaces for the description and configuration of the network to be attacked. All the notions and attributes that pertain to network and hosts, including the notions and attributes that describe firewalls, are attributed through this interface. This component is represented in paragraph 2.5.2.

Firewall model (implementation) component is used to determine the firewall's response to the action generated by the Hacker Agent. Each incoming message from the Hacker Agent that constitutes an attack action is entered into the firewall model, which is assigned to the entire network (in imitation of a network firewall) or (and) the attacked host (in imitation of a personal firewall). In the event of an attack being blocked by the firewall, the response message formed by the Network Agent contains only the information that the attack has been blocked by a specific firewall. The firewall model is described in paragraph 2.5.4.

Generator of the network's response to attack action is used for the generation of the network's and hosts' responses (messages) to attack actions. It is initialized through the corresponding function exported by the agent's kernel after the Hacker Agent has successfully overcome a firewall. The process of the response message generation by the Network Agent is considered in paragraph 2.5.4.

2.2. State-machine based descriptions of main components

The behaviors of both the Hacker Agent and the Network Agent specified on the basis of *state-machine models*, which are interpretations of behavior specified formally by use of formal grammar framework. The Hacker Agent acts on the basis of a complicated system of nested state machines. The state machine model of the Network Agent is represented by a single state machine.

The state machine model determines states, transitions between states, and conditions for such transitions. Each state corresponds to a description of actions that should be carried out when the state machine makes a transition into that state. These actions are initialized as the states of the state machines are processed. Actions are described through scripts in the MASDK Script Language.

The MASDK state machine model provides for four types of scripts, each of which is initiated from a certain point in the processing of the state machine's state:

1) *Entry script* – the type of scripts initialized if state machine transit in the respective state;

2) *Do Script* – a type of scripts initialized from the script after the entry script has been executed. If the script is initiated only at a single point from this state, then it doesn't matter if that point of initialization is the beginning, the middle, or the end of the script;

3) *Exit Script* – the type of scripts initialized on exiting a state;

4) *Action* – the type of scripts initialized on a transition from one state to another. This script is executed after the transition has been calculated (and the exit script has been executed).

The initialization (invocation) of a script may not be present.

Each script is uniquely identified in the state machines model by its name. Script names may coincide in agents of different classes, but are unique within a class.

To understand the principles of operation of scripts and state machines models in general, let us consider the notation of the script language used by agents. The description of scripts is found in the sections that deal with typical agent models (paragraphs 2.4.2 and 2.5.3). Examples of scripts are found in Appendixes 1 and 2.

As mentioned above, the script language serves to describe the agent's actions in terms of application domain and the global functions that are provided to the program realization of the agent by its Kernel. The script language is fairly simple, since it has to be used by an agent system developer but not by programmer. The language is pseudo object-oriented, since it uses object notation when addressing attributes of examples of notions and pre-determined functions, however, neither inheritance, encapsulation, nor polymorphism are supported. There is no need for the latter three. If a task to be solved by an agent is so complex that it requires object-oriented realization, MASDK provides for a mechanism for utilizing (plugging in) user components (for example, a probabilistic decision-making model component in Hacker Agent, or response generation component in Network Agent). These components are accessed through functions exported by the agent's kernel, and the functions can be initialized through scripts.

The agreements on usage of variables and constants are as follows:

- obvious declaration;
- beginning with the letter;
- only letters, digits and underlining marks may be used within variable or constant name;
- name could not contain a point;
- names are unique in "visibility" range.

The following types of variables are used (tab.2.2.1):

Tab.2.2.1. Types of variables

N	*Type*	*Description*
1	***bool***	TRUE, FALSE
2	***int***	Integer type (system dependent): –2,147,483,648 to 2,147,483,647
3	***double***	1.7E +/- 308 (15 digits)
4	***string***	String type, value must be placed in quotation marks

The agreements on declaration of variables are as follows.

Variables may be declared by two ways:

1) Variables of simple types (simple variables) are declared in the script, for example:

DECLARE string Str1;

2) Objects (instances of classes), and also global variables are created at the description of interpretation model.

The interpretation model is a superstructure above application domain ontology (component *DomainOntology*). It is a link between application domain ontology and scripts which use the ontology notions and their attributes as instances with their predetermined methods and attributes. The interpretation model is one of the MASDK components.

In the scripts of the state machine model only two global variables are used: *int dC* and *bool bX*.

The first variable sets the probability of terminal action success, and the second one – returned parameter (success or nonsuccess of the terminal action). Both variables are used as arguments of the global function *AttRandom* (of the Network Agent) which is described in section 2.5.3.

The agreements on declaration of constants are as follows.

Constants may be declared by two ways:

1) Constants of simple types (simple constants) are declared in a script, for example:

DECLARE string Str2 = "attacked host ip-address";
DECLARE bool Flag = TRUE;
DECLARE int i;

2) Global constants are created as properties of special objects at description of the interpretation model. They are not used in the prototype.

The following agreements on scope of script are used. Simple variables and constants have a scope of a script level. These variables and constants exist only in an operating time of the script of *Do Action* or *Entry Action*, and are deleted from memory under script operation completion.

Objects are global and accessible in all scripts of the model.

The operations of the Language are as follows:

(1) Mathematical operations;
(2) Comparison operations;
(3) Logical operations.

Mathematical operations are as follows (tab.2.2.2):

Tab.2.2.2. Mathematical operations

Operator	*Description*	*Operation priority*
^	Exponentiation	1
*	Multiplication	2
/	Division	2
\	Integer division	2

Operator	*Description*	*Operation priority*
+	Addition	3
-	Subtraction	3
-	Change of an operand sign	4
Mod	Residue of division	5

Comparison operations are as follows (tab.2.2.3):

Tab.2.2.3. Comparison operations

Operator	*Description*
=	Equal
<>	Not equal
<	Smaller
>	Large
<=	Smaller or equal
>=	Larger or equal

Logical operations are as follows (tab.2.2.4):

Tab.2.2.4. Logical operations

Operator	*Description*
NOT	Negation
AND	Logical multiplication
OR	Logical addition
XOR	Logical exclusion

The Language operators are as follows (tab.2.2.5):

Tab.2.2.5. Language operators

Operator	*Description / example*
;	Separator of operators
=	Assignment operator *ID = <Expr>;* *Expr* – simple variable, global variable, variable value, object attribute, object method, logical or arithmetic expression
Object method call	*Obj.Method([p1, p2,…]);* where *p1, p2* - arguments of a method; logical or arithmetic expressions may be arguments; *Obj* – direct instance of ontology concept; *Method* - predetermined method name or unique method name of the given class (the list of the predetermined class methods is described on the following table)
RETURN;	Interrupts script performance
BREAK;	Interrupts block performance; transfers the control to the first operator following the block
DECLARE *<value type> <name> [= <value> | <logical-and-arithmetic expression>];*	Declares a simple variable; *Value type* - one of 4 types of variables described above; *Logical-and-arithmetic expression* - the expression which consists of logic and (or) arithmetic expressions
SENDMESSAGE *(<msg name>);*	Sends the message to other agent (it is not used in the prototype); *msg name* – message name

IF (*<condition>****)***
THEN *<set of operators>*
ELSE *<set of operators>*
ENDIF;
	If operator;
	condition - logical expression (expression in which logical operations have the least performance priority), may consist of variable *bool*;
	set of operators - a sequence of the operators described in this table, divided by “;”

REPEAT *<set of operators>* ***UNTIL*** *(<condition>);*	Cycle operator with a postcondition

Operator	*Description / example*

WHILE(*<condition>)*
DO *<set of operators>*
END;
	Cycle operator with a precondition

CALLSCRIPT(*<script_name>);*	Call of a script with the name *script_name*
<global procedure name> *([<set of arguments>])*	Call of external procedure
AUTO(*<auto_name>);*	Initialization of the state machine with name *auto_name*

All operators in script are separated by symbol “;”.

In the MASDK script language the opening and closing block constructions are not used. This fact is stipulated by intention to simplify the script language as much as possible. In our opinion, scripts should be written not by a programmer, but by a model developer as well as by the problem originator.

The predetermined class methods are as follows (tab.2.2.6):

Tab.2.2.6. Predetermined class methods

Method	*Description / example*
Create();	Creates an instance of the notion
Exist(*<condition>****);***	Checks a presence of the class with the attributes specified in *condition*; in *condition* attributes of other classes, local and global variables, as well as constants may be used as operands of logic expression
Next();	Takes the next instance of the notion class; in case of success returns TRUE, and in case of failure – FALSE
Prev();	Takes the previous instance of the notion class; in case of success returns TRUE, and in case of failure – FALSE
First();	Takes the first instance of the notion class; in case of success returns TRUE, and in case of failure – FALSE; at the same time in the mapping database the cursor moves on the first record in the table corresponding to the specified notion
Last();	Takes the last instance of the notion class; in case of success returns TRUE, and in case of failure – FALSE; at the same time in the mapping database the cursor moves on the last record in the table corresponding to the specified notion
Delete(*<condition>****);***	Deletes instances of the notion class, which satisfy the specified *condition*; in the mapping database all records fitting the given class and satisfying the given *condition* are deleted
DeleteAll();	Deletes all instances of the notion class for which the given method is called; in the mapping database all records fitting the given class are deleted
Copy(*<class name >****);***	Copies an instance of the class *class name*; an instance (established by any of the predetermined method *Exist, Next, Prev, First,* or *Last*) on which the cursor points is a current one
CopyAll(*<class name>****);***	Copies all instances of the class *class name*

The comments are considered as any text between /* … */ or a part of a text line, beginning with a symbols //.

2.3. Component of the application domain ontology

This section contains detailed description of all notions of the application domain (hereinafter ontology) of the Attack Simulator realized in the component of the application domain ontology (*DomainOntology* component), as well as the description of the program component – ontology editor.

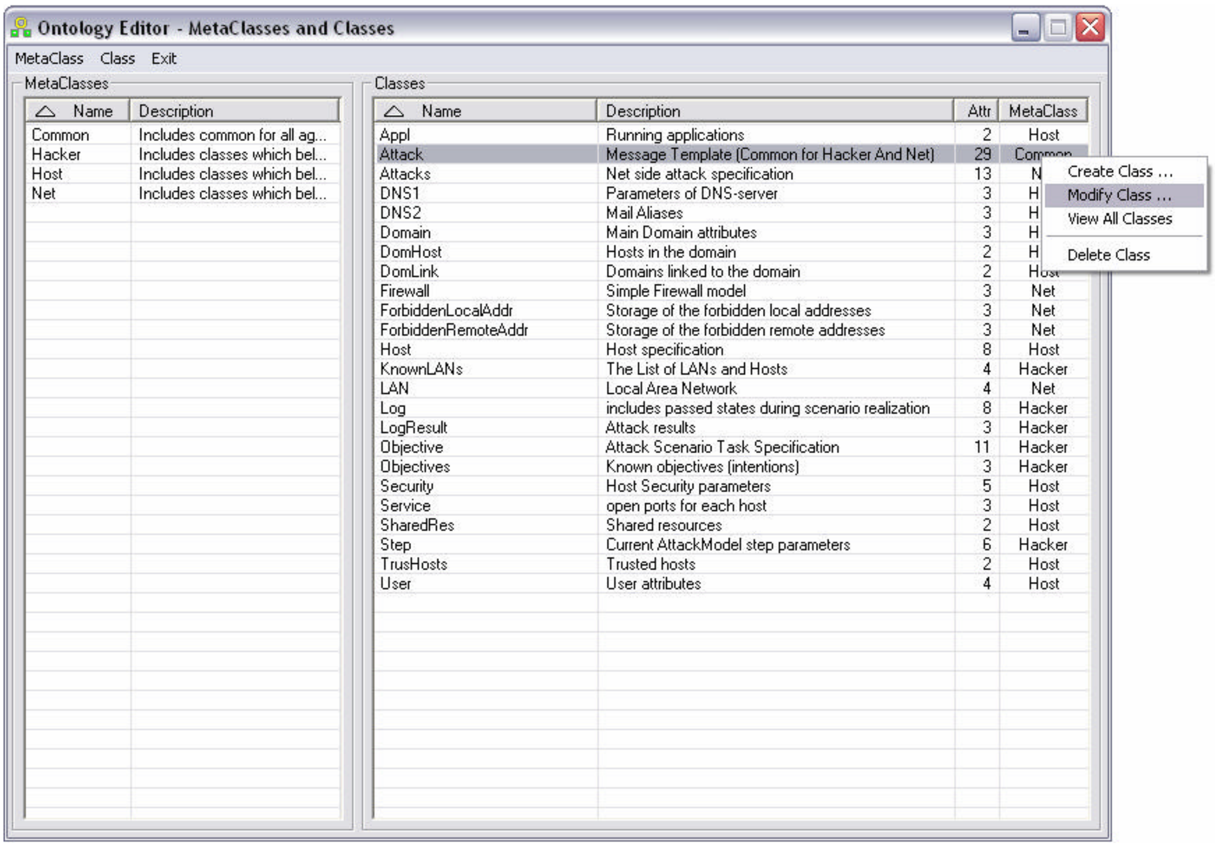
Ontology serves to specify and store notions, their attributes, and values of attributes of the application domain of computer network attacks modeling and simulation.

The software support of the ontology is realized in MASDK. Ontology is filled during the design stage through using the MASDK Ontology Editor. Classes, class attributes, and meta-classes that unify classes into groups are entered and modified through the ontology editor's user interface.

Screenshots of the Ontology Editor component of MASDK are shown in Fig.2.3.1 and Fig.2.3.2. In the window of the ontology editor (Fig.2.3.1), classes of the ontology notions of the prototype, as well as their categories (mataclasses), are shown. Fig.2.3.2 and Fig.2.3.3 present the modification process of the notion *Attack* and its attribute *ip*.

Meta-classes are used as classifiers of notions of the application domain, and are not used in the prototype's runtime.

Names of all meta-classes, notions, and their attributes can only contain digits and letters. They may only start with a letter and should not contain spaces or underscores. The names are case sensitive. These restrictions have to do with the usage of these names in the software implementation, specifically, in the script language, where a notion is a program class, and its attribute is an attribute of that class. The script language is considered in more detail in sections 2.2, 2.4.2 and 2.5.3.



Name	Description
Common	Includes common for all ag...
Hacker	Includes classes which bel...
Host	Includes classes which bel...
Net	Includes classes which bel...

Name	Description	Attr	MetaClass
Appl	Running applications	2	Host
Attack	Message Template (Common for Hacker And Net)	29	Common
Attacks	Net side attack specification	13	N
DNS1	Parameters of DNS-server	3	H
DNS2	Mail Aliases	3	H
Domain	Main Domain attributes	3	H
DomHost	Hosts in the domain	2	H
DomLink	Domains linked to the domain	2	Host
Firewall	Simple Firewall model	3	Net
ForbiddenLocalAddr	Storage of the forbidden local addresses	3	Net
ForbiddenRemoteAddr	Storage of the forbidden remote addresses	3	Net
Host	Host specification	8	Host
KnownLANs	The List of LANs and Hosts	4	Hacker
LAN	Local Area Network	4	Net
Log	includes passed states during scenario realization	8	Hacker
LogResult	Attack results	3	Hacker
Objective	Attack Scenario Task Specification	11	Hacker
Objectives	Known objectives (intentions)	3	Hacker
Security	Host Security parameters	5	Host
Service	open ports for each host	3	Host
SharedRes	Shared resources	2	Host
Step	Current AttackModel step parameters	6	Hacker
TrusHosts	Trusted hosts	2	Host
User	User attributes	4	Host

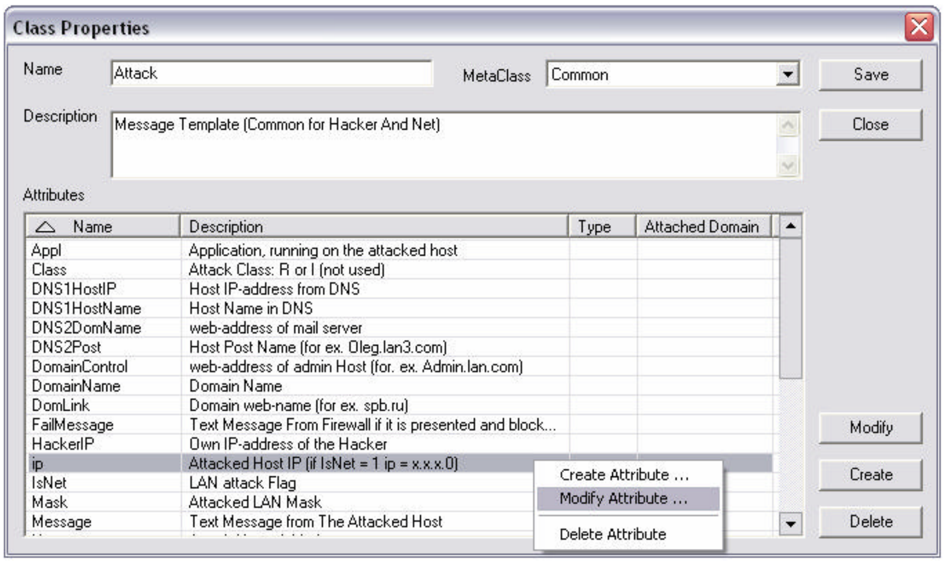
Fig.2.3.1. Example of ontology editor window supporting notion design and classification

Fig.2.3.2 shows a screenshot of "*Class Properties*" for the notion *Attack*. This notion is used in templates of messages exchanged by Network Agent *MainNet* and Hacker Agent *MainHack*, and it duplicates all the attributes of other notions, such as *Host*, *User*, *Domain*, etc., that are necessary for the communication.

In this window, the notions are being modified and classified; the window also enables further modification of the attributes list. A single attribute modification dialog is shown in Fig.2.3.3.

Attributes can belong to one of three classes:

- *Nominal* (line),
- *Double* (real floating point number; can be also used for storing integer values, dates and times), and
- *Unknown* (type is undefined for this attribute).



Class Properties

Name: Attack

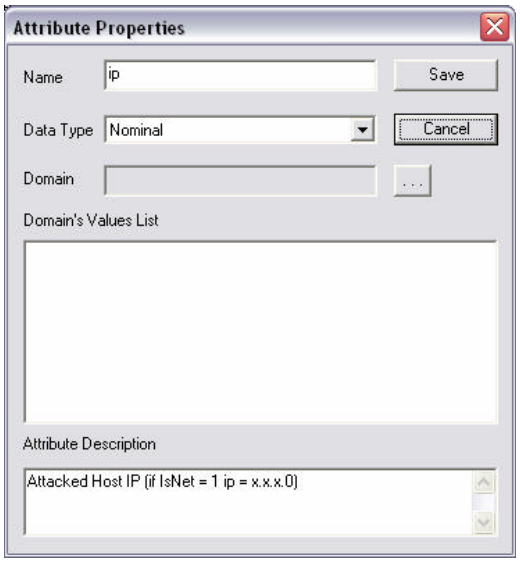
MetaClass: Common

Description: Message Template (Common for Hacker And Net)

Attributes

Name	Description	Type	Attached Domain
Appl	Application, running on the attacked host		
Class	Attack Class: R or I (not used)		
DNS1HostIP	Host IP-address from DNS		
DNS1HostName	Host Name in DNS		
DNS2DomName	web-address of mail server		
DNS2Post	Host Post Name (for ex. Oleg.lan3.com)		
DomainControl	web-address of admin Host (for. ex. Admin.lan.com)		
DomainName	Domain Name		
DomLink	Domain web-name (for ex. spb.ru)		
FailMessage	Text Message From Firewall if it is presented and block...		
HackerIP	Own IP-address of the Hacker		
ip	Attacked Host IP (if IsNet = 1 ip = x.x.x.0)		
IsNet	LAN attack Flag		
Mask	Attacked LAN Mask		
Message	Text Message from The Attacked Host		

Fig.2.3.2. Example of ontology editor window supporting notion modification and attribute value assignment



Attribute Properties

Name: ip

Data Type: Nominal

Domain:

Domain's Values List

Attribute Description: Attacked Host IP (if IsNet = 1 ip = x.x.x.0)

Fig.2.3.3. Example of ontology editor window supporting modification of notions' attributes

After the application domain ontology has been formed, for each of the agent classes *NetAgent* and *Hacker* that contain one agent *MainNet* and *MainHack* each, respectively, a set of ontology notions is specified that will be used for the operation of the agents of that class.

One should note, however, that the MASDK environment allows for creation of any number of both agent classes and agents within a class, as well as any number of ontologies (in the prototype, only one ontology is used).

In specifying the set of notions for each agent class, it is necessary to take into account the fact that if agents of two classes perform their actions in collaboration, they should exchange information in the manner understandable by both agents. This implies that notions that participate in the communication between agents of two different classes should be present in both agent classes' notion lists.

Let us further consider the general notions of the application domain ontology.

Descriptions of the fragments of ontology used by agents *MainHack* and *MainNet* are found in sections 2.4.1 and 2.5.1 respectively.

Fig.2.3.4 shows a diagram of application domain ontology classes.

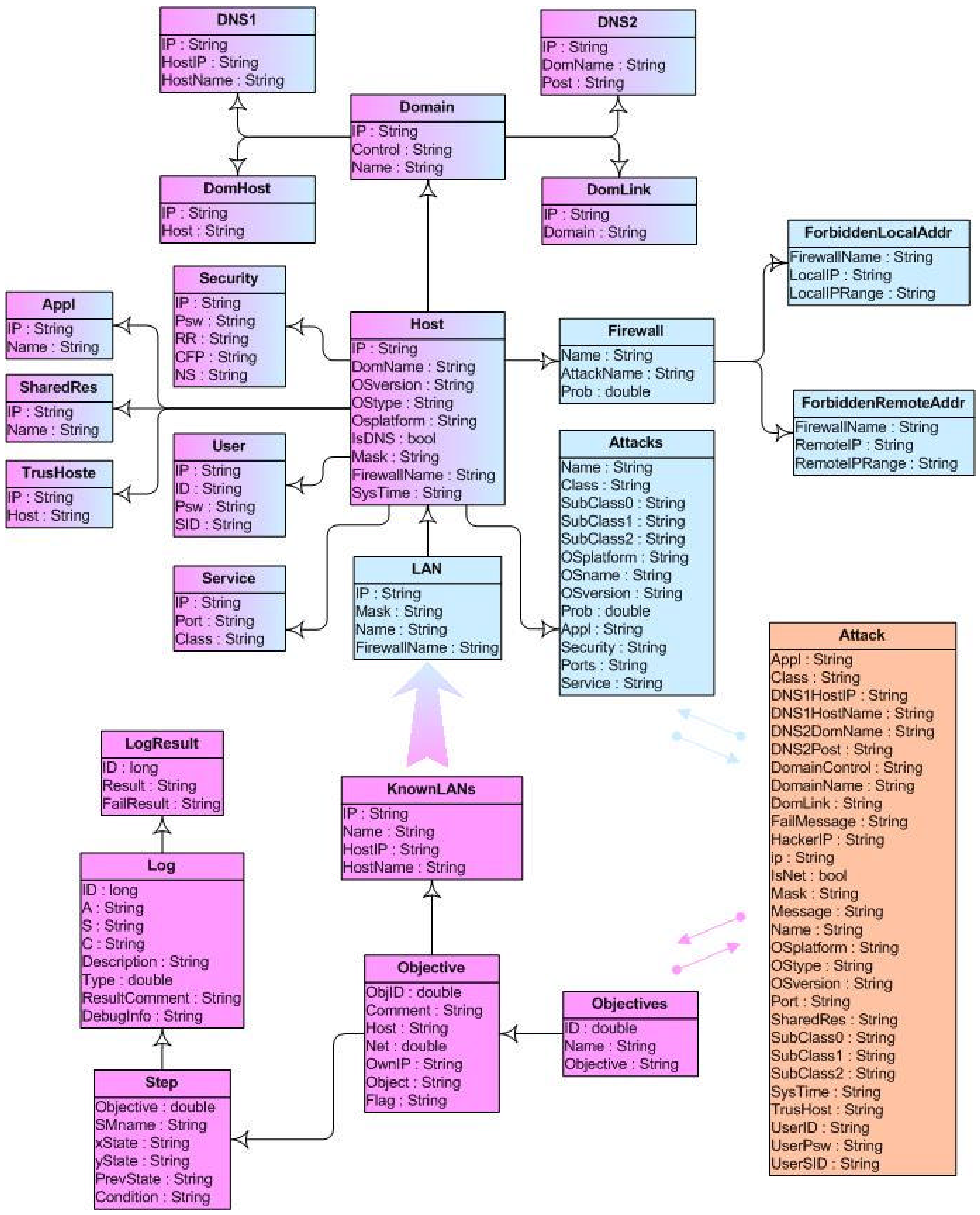


Fig.2.3.4. Diagram of the application domain ontology classes

1. The notion ***Appl*** serves to store the names of applications running on the attacked host. It has the following attributes:

- ***IP*** – host's IP address;
- ***Name*** – name of the running application.

2. The notion ***Attack*** serves to ensure communication between agents *MainHack* and *MainNet*. This notion has the following attributes most of which are activated in attack generation (star symbol denotes attributes mandatory for any attack; the direction of message exchange between agents is showed in parentheses, where the first agent determines attribute value, and the second one uses this value):

- ***Appl*** – application running on the host (MainNet → MainHack);
- ***Class*** – generalized attack class (reconnaissance (R) of Implantation and threat realization (I));
- ***DNS1HostIP*** – host's IP address taken from the attacked server (one message per every IP address taken from the DNS server); sent together with *DNS1HostName* (MainNet → MainHack);
- ***DNS1HostName*** – host's name taken from the host, if it is a DNS server (MainNet → MainHack);
- ***DNS2DomName*** – name of domain whose server is being attacked (MainNet → MainHack);
- ***DNS2Post*** – host's mail address taken from the attacked server; sent together with *DNS1HostIP* (MainNet → MainHack);
- ***DomainControl*** – DNS address of the attacked DNS server (MainNet → MainHack);
- ***DomainName*** – DNS address of network (MainNet → MainHack);
- ***DomLink*** – DNS address of the trusted network for the attacked DNS server (MainNet → MainHack);
- ***FailMessage*** – text message sent by the network-agent *MainNet* in case the attack is blocked on the network side or by either a network or a local firewall; the message contains the following information: name of firewall, IP address, content: the attack is detected and blocked, or the sender's (i.e. hacker's) IP address is blocked (MainNet → MainHack);
- ***HackerIP*** – hacker's IP address (MainHack → MainNet)*;
- ***ip*** – IP address of the attacked host or network (MainHack → MainNet)*;
- ***IsNet*** – parameter that shows the agent of the attacked network that the attack is a network attack (directed to all hosts of the network) (MainHack → MainNet)*;
- ***Mask*** – attacked subnet mask (MainNet → MainHack);
- ***Message*** – text message (MainNet → MainHack);
- ***Name*** – name (abbreviation) of the attack action (MainHack → MainNet)*;
- ***OSplatform*** – attacked host's OS platform (Windows or Unix) (MainNet → MainHack);
- ***OStype*** – OS type (98, ME, XP, SunOS, Linux Mandrace, etc.) (MainNet → MainHack);
- ***OSversion*** – OS version (e.g., SE, SP1, 1.4.1, 7) (MainNet → MainHack);
- ***Port*** – number of open port on the attacked host; a list of ports is formed for each of the attacked hosts and sent in a single message (MainNet → MainHack);
- ***SharedRes*** – name of open resource; a list of resources is formed for each of the attacked hosts and sent in a single message (MainNet → MainHack);
- ***SubClass0*** – 0th level subclass of the attack action specified in the attribute *Name* (MainHack → MainNet);
- ***SubClass1*** – 1st level subclass of the attack action specified in the attribute *Name* (MainHack → MainNet);
- ***SubClass2*** – 2nd level subclass of the attack action specified in the attribute *Name* (MainHack → MainNet);
- ***SysTime*** – current time on the attacked host (to a millisecond) (MainNet → MainHack);
- ***TrusHost*** – trusted host's IP address; a list of trusted hosts (if available) is formed for each of the attacked hosts and sent in a single packet (MainNet → MainHack);
- ***UserID*** – user name (from the attacked host's account); all obtained user names are sent in separate packets for each of the attacked hosts (in case of a network attack); sent together with *UserPsw*, while each of the attributes may not be present (MainNet → MainHack);
- ***UserPsw*** – user password (from the attacked host's account); sent together with *UserID*, while each of the attributes may not be present (MainNet → MainHack);

- ***UserSID*** – user's SID, obtained by hacker only if the DNS server attack is successful (MainNet → MainHack).

3. The notion ***Attacks*** determines the knowledge of the agent *MainNet* about network attacks and has the following attributes (star symbol denotes attributes mandatory for any attack):

- ***Appl*** – list of applications running on the host under attack with the name *Name* (hereinafter for attributes of this notion – provided that all the rest of the conditions specified in the example of this notion are met);
- ***Class*** – class to which the attack called *Name* belongs*;
- ***Name*** – name of attack*;
- ***OSname*** – types of OS for which the attack called *Name* is performed;
- ***OSplatform*** – OS platforms for which the attack called *Name* is performed;
- ***OSversion*** – OS versions for which the attack called *Name* is performed;
- ***Ports*** – open ports for which the attack called *Name* is performed;
- ***Prob*** – probability of attack called *Name* in case all other conditions are met*;
- ***Security*** – security parameters independent of firewall attributes;
- ***Service*** – network services and/or protocols (protocols stack), for which the attack called *Name* is performed;
- ***SubClass0*** – 0th level subclass of attacked action called *Name**;
- ***SubClass1*** – 1st level subclass of attacked action called *Name**;
- ***SubClass2*** – 2nd level subclass of attacked action called *Name**.

4. The notion ***DNS1*** has the following attributes:

- ***HostName*** – client-host name;
- ***IP*** – IP address of network domain;
- ***HostIP*** – client-host's IP address.

5. The notion ***DNS2*** has the following attributes:

- ***DomName*** – mail server name;
- ***IP*** – network mail server's IP address;
- ***Post*** – mail account.

6. The notion ***Domain*** has the following attributes:

- ***Control*** – domain-host name;
- ***IP*** – domain's IP address;
- ***Name*** – domain's network name.

7. The notion ***DomHost*** has the following attributes:

- ***Host*** – host name;
- ***IP*** – host's IP address.

8. The notion ***DomLink*** has the following atributes:

- ***Domain*** – linked domain's network name;
- ***IP*** – domain's IP address.

9. The notion ***Firewall*** has the following attributes:

- ***Name*** – firewall name;
- ***AttackName*** – abbreviated name of the attack that is present in the firewall knowledge base;
- ***Prob*** – probability of the attack called *AttackName*.

10. The notion ***ForbiddenLocalAddr*** has the following attributes:

- ***FirewallName*** – firewall name;

- ***LocalIP*** – forbidden local address of the firewall called *FirewallName*;
- ***LocalIPRange*** – range of forbidden local addresses for the firewall called *FirewallName*.

11. The notion ***ForbiddenRemoteAddr*** has the following attributes:

- ***FirewallName*** – name of firewall;
- ***RemoteIP*** – forbidden remote address for the firewall called *FirewallName*;
- ***RemoteIPRange*** – range of forbidden remote addresses for the firewall called *FirewallName*.

12. The notion ***Host*** has the following attributes:

- ***DomName*** – host's domain name;
- ***IP*** – host's IP address;
- ***OSVersion*** – OS version;
- ***Mask*** – host's subnet mask;
- ***SysTime*** – host's system time (further formed dynamically);
- ***IsDNS*** – attribute that indicates whether the host is a domain server;
- ***OSType*** – OS type;
- ***OSPlatform*** – OS platform (presently two platform, Windows and Unix, are used);
- ***FirewallName*** –firewall name (may be absent).

13. The notion ***KnownLANs*** determines hacker's knowledge about networks that have been attacked before or will be attacked in the future. This notion has the following attributes:

- ***HostIP*** – host's IP address;
- ***HostName*** – host's network name;
- ***IP*** – network's address;
- ***Name*** – workgroup or domain name.

14. The notion ***LAN*** determines the network's knowledge of itself and has the following attributes:

- ***HostIP*** – host's IP address;
- ***HostName*** – host's network name;
- ***IP*** – network's address;
- ***Name*** – workgroup or domain name;
- ***FirewallName*** – name of network firewall (may be absent).

15. The notion ***Log*** stores the attack route in terms of state machine model and the obtained results. This notion has the following attributes:

- ***A*** – state machine name;
- ***DebugInfo*** – auxiliary information that determines the chain leading to the current state starting from automaton A ("Network attack");
- ***Description*** – description of the state machine's state (except for the intermediate states); if the state is terminal, then the action description is specified; if it is non-terminal, then the description of attack class (0th, 1st or 2nd level) is recorded;
- ***ID*** – a unique number identifying the past state of a state machine (this attribute is connected to the attribute *ID* of the notion *LogResult*);
- ***ResultComment*** – the description of the result that can be obtained in the used state *S* (if that state is terminal);
- ***S*** – the used state of a state machine;
- ***Type*** – state machine's state type: 0 – intermediate, 1 – terminal, in which the attack actions are performed, 2 – non-terminal (leading to other states);
- ***C*** – additional comments.

16. The notion ***LogResult*** has the following attributes:

- ***ID*** – number of used terminal state that has led to success of the attacker;

- ***Result*** – information received from the host or message about the successful attack in the terminal state with the number *ID*;
- ***FailResult*** – information received from the attacked network in case the attacked is blocked by a firewall; this attribute corresponds to the attribute *FailMessage* of the notion *Attack*.

17. The notion ***Objective*** stores the description of malefactor's intention for one attack scenario generation session. This notion has the following attributes:

- ***Comment*** – description of intention;
- ***Flag*** – system parameter;
- ***Host*** – IP address of the attacked host (network);
- ***LowLevel*** – parameter for inclusion of low-level network traffic generation in the attack scenario;
- ***Net*** – parameter that shows whether the attack is a network attack or not;
- ***Object*** – attack object (corresponds to the optional variable in attack goal specification);
- ***ObjID*** – number of intention; the attacker's intentions are described in more detail in paragraph 2.4.3;
- ***OwnIP*** – hacker's own IP address.
- ***PSWfile*** – path to the password file (dictionary of most frequently used words);
- ***SaveLog*** – logical parameter (with values "true" (1) and "false" (0)); if this parameter is true, then the attack scenario to be executed will use the information accumulated in the previous attacks (if there any have been realized), and traces and logs will also be saved;
- ***ShamIP*** – false (spoofed) IP address, used to specify the address of the host – source of the message; specifying a false IP address is used by the hacker for substituting its own address by another (for example, the address for the target's trusted host).

18. The notion ***Objectives*** stores descriptions of all intentions of the attacker realized in the prototype, and has the following attributes:

- ***ID*** – intention's unique number;
- ***Name*** – abbreviated name of the intention;
- ***Objective*** – description of the intention.

19. The notion ***Security*** has the following attributes:

- ***NS*** –Null Sessions parameter (used if *Host.OSPlatform* = *Windows*);
- ***IP*** – host's IP address;
- ***CFP*** **–** parameter that shows the presence or the absence of shared files and printers on the host;
- ***Psw*** – parameter that shows the presence or the absence of the password to enter the system;
- ***RR*** – parameter that shows the presence or the absence of Remote Registry on the host.

20. The notion ***Service*** has the following attributes:

- ***Class*** – protocol name (TCP, UDP, ICMP, etc.);
- ***IP*** – host's IP address;
- ***Port*** – open port number.

21. The notion ***SharedRes*** has the following attributes:

- ***IP*** – host's IP address;
- ***Name*** – path to shared resource.

22. The notion ***Step*** has the following attributes:

- ***Condition*** – condition of transition at the current step from the state *xState* into the state *yState*;

- ***Objective*** – hacker's intention realized at the current step (in the current realization of the prototype, intention is not changed in the single attack scenario generation session);
- ***PrevState*** – the state before the state *xState*;
- ***SMname*** – state machine's name;
- ***xState*** – state machine's current state;
- ***yState*** – the state, into which the state machine model makes the transition at the current step.

23. The notion ***TrusHosts*** has the following attributes:
- ***Host*** – attacked host's IP address;
- ***IP*** – trusted host's IP address.

24. The notion ***User*** has the following attributes:
- ***SID*** – user's security identifier;
- ***IP*** – host's IP address;
- ***ID*** – user name;
- ***Psw*** – user password to enter the system.

One host may have several users with individual accounts.

2.4. Generic Hacker Agent

As noted above, the system provides for two classes of agents: Hacker agent (class *Hacker*) and Network Agent (class *NetAgent*). In the prototype, one agent has been realized for each class: agent *MainHack* for the *Hacker* class, and agent *MainNet* for the *NetAgent* class.

A generic Hacker Agent consists of both a set of components realized within the MASDK environment, and external components developed in MS Visual C++.

The following are the main components of a generic Hacker Agent:

- Fragment of general ontology of computer network attack generation application domain, that is used by Hacker Agent in its operation;
- State machine model that realizes Hacker Agent's behavior scenarios;
- Set of scripts directing the Hacker Agent's behavior in each of the states of the state machine model and connecting the program modules of the Hacker Agent with the rest of its components realized in the MASDK environment;
- Set of program modules that realize the necessary functions and user interfaces created outside of MASDK. Program modules can be classified onto three groups:
  - Modules responsible for attack goal specification;
  - Modules that calculate the probabilistic model of hacker's behavior;
  - Modules generating network traffic.

2.4.1. Fragment of the ontology used by Hacker Agent

A detailed description of the application domain ontology of computer network attack generation is found in section 2.3. Let us specify the notions used exclusively by Hacker Agent. Let us consider these notions by breaking them up into three categories by the function of notions.

1. *Notions used in the attack goal specification process before the generation of its scenario*:
   1) ***Objectives***. The notion that stores the list of all attacker's intentions. At present, 12 high-level objectives are represented in the prototype. Intention is the most important attribute specified in the process of attack goal specification. Attack specification is considered in greater detail in 2.4.3.
   2) ***Objective***. The notion that stores the intention chosen at the attack specification stage. This notion has the following attributes:
      - ***ObjID*** – number of one of the intentions of the notion *Objectives;*
      - ***ip –*** IP address of the attacked host (network);
      - ***IsNet*** – marker that shows if the attack is a network attack;

- ***OwnIP*** – own IP address;
- ***Object*** – object of attack.

3) ***KnownLANs***. An auxiliary notion that participates in the attack task specification. It stores the list of attacked networks with the names and IP addresses of each network's hosts. The IP address of the host or network that will be attacked (the attribute *Objective.ip*) is chosen from that list and is a mandatory attribute in attack specification.

At the attack task specification stage, one can also mark the additional information that the hacker has about each of the hosts from the list *KnownLANs* (description of the user interface for specifying the field *Known Information* is found in 2.4.3). This information will automatically be taken into account in the attack generation. Thus, some or all notions from the second category are used. Determination of these parameters is not necessary for the functioning of the prototype.

2. *Notions used for storing the information received through attack deployment from the attacked network, obtained through the previous attacks, or specified as input data at the attack task specification*:
   1) ***Host***;
   2) ***TrusHosts***;
   3) ***SharedRes***;
   4) ***Domain***;
   5) ***DomHost***;
   6) ***DomLink***;
   7) ***DNS1***;
   8) ***DNS2***;
   9) ***User***;
   10) ***Security***;
   11) ***Service***;
   12) ***Appl***.

   These notions are described in paragraph 2.3. They are used to store information about hosts both on the Hacker Agent side and the attacked Network Agent side.

3. *Notions used by the Hacker Agent in the process of attack generation for storing the decision made at the current step and for recording the attack log*:
   1) ***Step***. This notion is filled at each step of the state machine model's operation in the process of deploying and implementing the attack scenario. It includes the following attributes:
      - ***Condition*** – condition for the transition from the state *xState* into the state *yState* at the current step. This condition is formed directly in the state machine model itself during the design time. On the realization stage, all conditions for each of the transitions are already specified where necessary. This attribute only stores the order number of the condition from the list of transition conditions (from state *xState* into state *yState* in the state machine called *SMname*);
      - ***Objective*** – intention realized by Hacker Agent at the current step (in the current version of the prototype it does not change within a single attack scenario generation session);
      - ***PrevState*** – state that precedes the current state *xState* of the Hacker Agent. This attribute is analyzed in the calculation of the further actions in some attacks of class R ("Reconnaissance");
      - ***SMname*** – name of the state machine realized by Hacker Agent at the current step;
      - ***xState*** – current state of the state machine;
      - ***yState*** – state into which the state machine makes the transition at the current step.
   2) ***Log***. This notion is responsible for logging the attack. Each record of the log is a piece of information about the used state of the state machine; it includes the following data (attributes):

- ***S*** – name of the used state;
- ***ID*** – the unique number of the used state of the state machine (this attributed is connected to the attribute *ID* of the notion *LogResult*);
- ***Description*** – description of the state (with the exception of intermediate states); if the state is terminal, then action description is specified, if it is non-terminal, then the description of attack class (0th, 1st or 2nd level) is recorded;
- ***A*** – name of the parent state machine, i.e. the state machine that generated the current action with the name ***S***;
- ***DebugInfo*** – auxiliary information that specifies the chain of transitions leading to the current state, starting from the state machine A ("Network attack"). For example, for the state *NS* that provides for the attack action "Collection of additional information from DNS-server", *DebugInfo* will be as follows: *A => R => CI => NS*, where *CI* (Collecting Additional Information) is a state machine (specified by attribute *A*), in which the action *NS* is generated. The state machine model of Hacker Agent operation is represented in greater detail in section 2.4.2;
- ***Type*** – type of state of the state machine: 0 – intermediate, 1 – terminal, in which the attack actions are performed, 2 – non-terminal (leading to other states). For example, the abovementioned state *NS* is terminal (*Type*=1), for in it the attacker's actions are carried out, and the state *CI* of the state machine *R* is non-terminal (*Type*=2), for in it no action are carried out to implement the threat and only other states are generated;
- ***C*** – additional comments;

3) ***LogResult***. Stores information obtained from the attacked network in each used state with the unique identifier *ID*. If the attacker's action were unsuccessful in the used state, that state is not recorded in *LogResult*, and is only recorded in the example of the notion *Log*. If the attacker's actions in the used state were blocked by a firewall, then the corresponding message is stored in the attribute *FailResult*. This notion is directly related to the attack implementation visualization component (considered in section 2.4.6).

4. *Notions used for exchange of information between Hacker Agent and attacked Network Agent through their attributes*:
   1) **Attack**.
   This notion is considered in great detail in section 2.3. Here, we will list those attributes of this notion that need to be determined in sending of messages by Hacker Agent to the Network Agent:
      - ***Name*** – name (abbreviation) of attack action;
      - ***ip*** – IP address of attacked host or network;
      - ***IsNet*** – parameter that indicates to the attacked Network Agent that the attack is a network attack (*IsNet*=1), or is directed at a separate host (*IsNet*=0). If it is a network attack, it will include all hosts of that subnet, and the IP address will look like this: x.x.x.0;
      - ***HackerIP*** – hacker's IP address; is necessarily sent to the Network Agent, though may be false;
      - ***SubClass0***, ***SubClass1***, ***SubClass2*** – attributes that tell the Network Agent the subclass of the hacker's attack action. These attributes are necessary to eliminate ambiguous interpretation of attack by Network Agent when only the abbreviation of the attack, specified by the attribute *Name*, is used. The same attack (attack action) may be present in different attack subclasses, but may have different implementations and response models.

2.4.2. State machines model of the Hacker Agent operation

In this paragraph the Hacker Agent states machine model is considered, a general structure of nested state machines (functional and communicational) is defined, and Hacker Agent actions (being fulfilled inside state machines) are determined by scripts language.

The state machine based model of Hacker Agent operation is realized in the component *AttackModel*. This component is used to specify the set of different classes of computer network attacks in the form of a family of state machines.

The following two principles have been embedded into the component *AttackModel* in its development:

1. The starting point of forming a specific attack scenario is the attack task specification or a top-level attack goal, with the "*Malefactor's intention*" being the most meaningful element of the attack task specification.

Twelve different malefactor's intentions in realizing computer network attacks have been identified that can be included in the attack task specification:

- Identification of Hosts (IH);
- Identification of Services (IS);
- Identification of Operating system (IO);
- Resource Enumeration (RE);
- Users and groups Enumeration (UE);
- Applications and Banners Enumeration (ABE);
- Gaining Access to Resources (GAR);
- Escalating Privilege (EP);
- Confidentiality Violation Realization (CVR);
- Integrity Violation Realization (IVR);
- Availability Violation Realization (AVR);
- Creating Back Doors (CBD).

The first six intentions are associated with the top-level intention "*Reconnaissance*" (*R*), and the other six – with the top-level intention "*Implantation and threat realization*" (*I*).

Top-level intentions may alternate and repeat themselves. This is embedded in the state machine *A (Network Attack)* that belongs to the topmost level.

2. The entire set of attack scenarios can be achieved through the sequential decomposition of the malefactor's intentions into sub-intentions and different classes of attacks (attack actions). This process of decomposition is fulfilled until concrete actions of the attacker in relation to the attacked network (host) are represented on the lowest level. This level, depending on the goals of attack modeling, can be specified both by the symbols of the attacker's actions and the concrete network packets, OS commands, etc.

After forming the hacker's action the corresponding communicational state machine starts to work. This state machine transmits the hacker's action to the attacked Network Agent as a message or a set of messages.

The sequence of the attacker's actions, derived from the initial attack task specification as well as the logic of the functioning of the family of state machines and the used probabilistic models, is the concrete realization of a class of possible attack scenario variants.

The basic functioning unit of the component *AttackModel* is a state machine.

All set of state machines consists of two classes:

- Functional state machines, and
- Communicational state machines.

Functional state machines make decisions about the Hacker Agent's next steps and prepare attack actions. Communicational state machines transmit prepared messages to the Network Agent and process received messages.

The list of constructed state machines is given in Tab.2.4.1. Their total amount is 48, including 33 functional and 15 communicational state machines. The entire set of state machines forms a hierarchy of interrelated state machines. This hierarchy is represented in Fig.2.4.1 as the interaction diagram.

In the diagram, the connections between state machines designated by arrows point out the fact that the upper state machine creates the bottom one (under the arrow) in one of its states. Connections designated by the square with a "P" inside designate the delegation of control to the state machine that has the square. The delegation may be done both ways, for example, like between the state machines *R* and *I*.

Tab.2.4.1. Description of state machines

Symbol	Description
A	Network Attack
ACE	Access Commands Execution and connection closing
ABE	Applications and Banners Enumeration
ABE_MSG	Communication automat ABE
AVR	Availability Violation Realization
CBD	Creating Back Doors
CBD_MSG	Communication automat CBD
CI	Collecting of additional Information
CI_MSG	Communication automat CI
CSS	Combined IP spoofing on SunOS v.1.4.x
CT	Covering Tracks
CT_MSG	Communication automat CT
CVR	Confidentiality Violation Realization
CVR_MSG	Communication automat CVR
DCSR	Direct Connection to Shared Resource
DS	Denial of Service (DoS) attack
EKV	Exploitation of Known server application Vu lnerabilities
ENS	Enumerating NetBIOS Shares
ENS_MSG	Communication automat ENS
EP	Escalating Privilege
EP_MSG	Communication automat EP
GAD	Gaining Additional Data
GAD_MSG	Communication automat GAD
GAR	Getting Access to Resources
GAR_MSG	Communication automat GAR
I	Implantation and threat realization
IAUS	Identifying Accounts with user2sid/sid2user
IBSD	Installation of Backdoor Server Daemons and Trojans and access to a host
IH	Identification of Hosts
IH_MSG	Communication automat IH
IO	Identification of Operating System
IO_MSG	Communication automat IO
IS	Identification of Services
IVR	Integrity Violation Realization
IVR_MSG	Communication automat IVR
PSA	Password Stealing Attack and access to a host
R	Reconnaissance
RCE	Registry Content Enumeration
RE	Resource Enumeration
RE_MSG	Communication automat RE
RRM	Remote Registry Manipulation and access to a host
SPIH	Port Scanning during Host Identification
SPIS	Port Scanning during Identification of Services
SPIS_MSG	Communication automat SPIS
TR	Threat Realization
UE	Users and groups Enumeration
UE_MSG	Communication automat UE
UFPS	Use of File and Print Sharing

The latest state machines in sequences are communicational state machines.

The delegation occurs on the level of states of the state machine that has created these state machines. In other words, the delegation of control on the state machine level (for example, between state machines *R* and *I*) is a transition between the states of the state machine of a higher level (in this case, state machine *A*), whose states *R* and *I* call the corresponding state machines *R* and *I*.

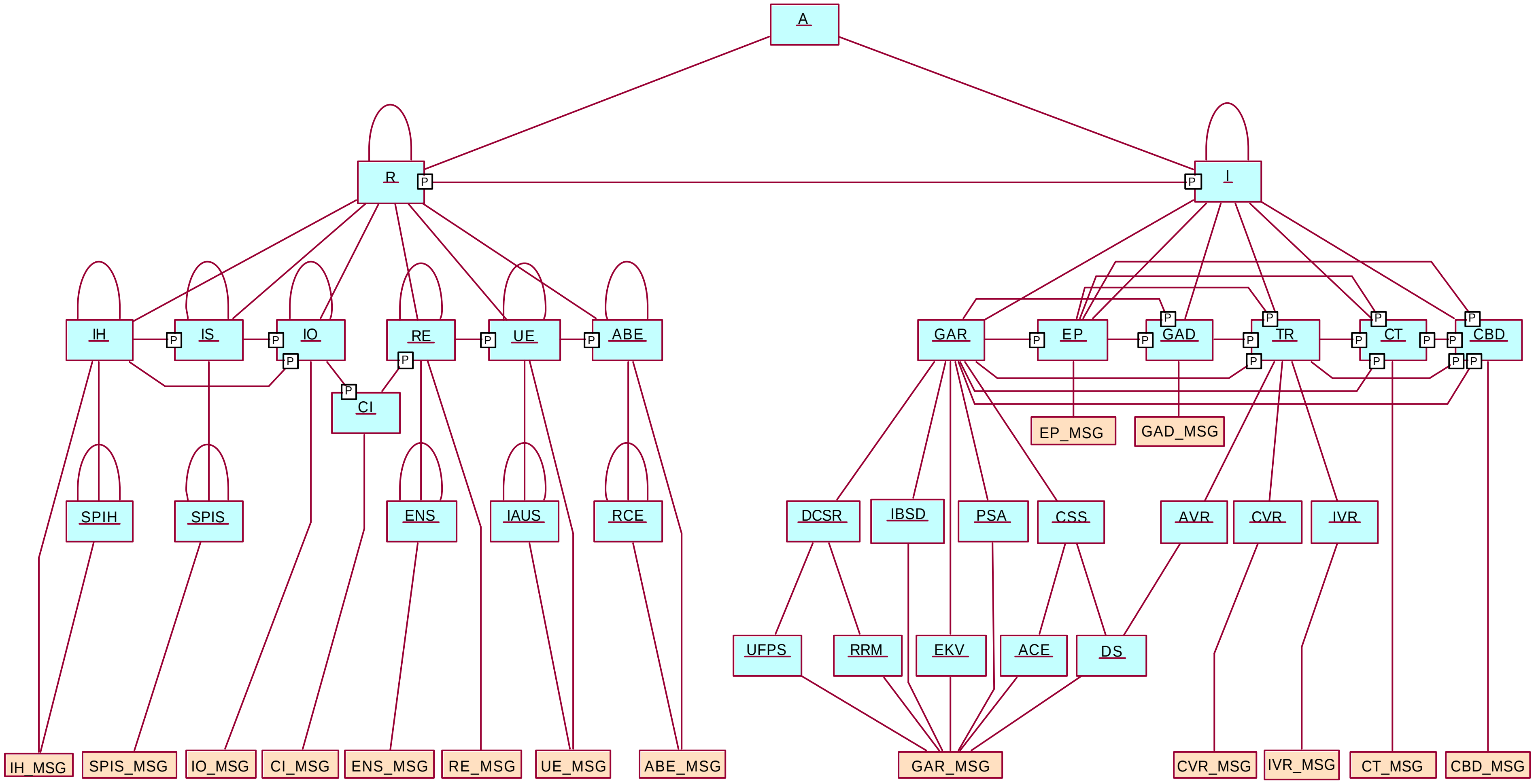


Fig.2.4.1. State machines interaction diagram

In case of two-way delegation, like in case of *R* and *I*, in the parent state machine *A*, two transitions occur between the states *R* and *I*: *R* => *I* and *I* => *R*.

States and transitions between states are the main elements of a state machine. Each state machine has a single entry point and exit point.

The states are divided by their functional load into three types:

- non-terminal,
- terminal, and
- intermediate.

Non-terminal states are the ones in which the state machines are created (hereinafter the created state machines are called nested state machines).

Terminal states are the ones in which concrete actions are performed to realize the attack on a computer network (host).

Intermediate states are auxiliary linking and differentiating nodes, in which no actions to realize attacks are performed and no nested state machines are created.

To each state machine, its own transition table is mapped. It is used for transition under the current condition from one state into another. According to the Booch notation, only one unconditional transition into the first meaningful state is possible from the state machine's entry point. In order to observe this rule in a case when several transitions have to be specified from the entry point, depending on the essence of the attack, an auxiliary intermediate state is created as the first meaningful state, from which several transitions are realized.

Three kinds of action are possible in each state of the state machine:

- *Entry Action*,
- *Do Action*,
- *Exit Action*.

Entry Action is performed upon entering the state, *Do Action* – within the state, and *Exit Action* – upon exiting the state.

Actions are specified as collections of scripts performed by the component *Engine*. The scripts describe the logic of a state machine's behavior in each separate state (flexible behavior). MAS DK Script language is used to describe behavior (for scripts representation).

A script interacts with the application domain ontology component by calling the notions (entities) and their attributes, calls the predefined and unique methods of each of the functions and the global functions, and also calls other scripts.

Scripts that describe the actions *Do Action* essentially form the logic of attack scenarios implementation. Through complicating these scripts, the functionality of the attack simulator prototype is developed, including the realization of the functions of analyzing the information received from the attacked network (host) and the interaction with it.

Scripts that describe the actions *Exit Action* form the log of the realization trace of the current attack scenario and the creation of nested state machines in non-terminal states.

The state machines model component is initialized in the state *A* of the basic state machine *A*, and finishes its work in the state *End* of the same state machine. Upon completion of its work, each state machine delegates control to the state of the state machine from which the machine that has just finished work was called (created), and thus control is always returned to the basic state machine.

The examples of state machines are represented in Appendix 1. These are the following state machines: functional state machines A (Network attack), *R* (Reconnaissance), *I* (Implantation and threat realization), *IH* (Identification of Hosts), *SPIH* (Port Scanning) and communicational state machine *IH_MSG*. The more detailed descriptions of all state machines (except for communicational state machines) nave been presented in *the Interim Report #3* [IntRep#3].

The scheme of description of each state machine is as follows:

1. *Identifier of the node to which the state machine corresponds*. The identifier reflects the nestedness level in relation to the basic state machine, and the last number represents the sequence number of the state machine's state from which the state machine in question is created.

2. *State machine diagram*. This diagram depicts the state machine's rigid behavior. Here, all the states of the state machine are listed (terminal, non-terminal, intermediate), as well as the transitions between them.

3. *Main parameters of the state machine*. Here the conceptual part of the state machine diagram is represented as a table, including the intentions this state machine realizes.

4. *Parameters of transitions*. Here, all the transitions between the states of the state machine are recorded as a table, where:

N is the sequential number of the transition;

CS (Current State) – the state from which the transition is made;

Script Name – name of script (scripts) executed on this transaction (scripts can be common for several transactions; as a rule, these transitions form a separate probabilistic group);

NS (Next State) – the state into which the transition is made;

Cond (Condition) – the sequential number of transition condition (may be absent); transitions with the same CS and NS but with different conditions form different probabilistic groups;

Intentions – section in which the probability of each transition being chosen (in its probabilistic group) is stated depending on the intention; the number of intentions actualized in the state machine is recorded in item 3 of the state machine description outline (main parameters of the state machine); in a case when the intention is not considered in the state machine, the column corresponding to that intention will contain zeros; in the intentions section, the line with probability recalculation factors Kj ($j \in [1..12]$) determined for each transition and depending on intentions is highlighted with yellow color (these factors may be absent, which implies Kj = 1, where j is the number of intention); all probabilities within the probabilistic group are recalculated after the choice of a transition with factor $k \neq 1$ within that group, in order to increase (decrease) the probability of this and all the other transitions within the probabilistic group being chosen; probabilistic group is a set of transitions whose total probability of being chosen by each of the intentions equals 1 (both at the start of the component's operation and after the recalculation of probabilities); the probabilities recalculation mechanism, as well as the probabilistic model and its implementation are considered in greater detail in paragraph 2.4.4. The group of transitions may consist of one transition, and in this case, this transition is considered unconditional – the probability P of this transition being chosen equals 1.

5. *Transition conditions*. Here, each logical condition specified in the section *Cond* among parameters of transitions. This logical condition is present in the scripts that describe the actions *Do Action* assigned to the states from which the transitions are performed by the specified condition numbers (at least one transition is performed). If for all the transitions from the given state the section *Cond* is empty, this means that only unconditional transitions and only based on the choice by probability can be performed into all the states in the table of transitions from the current state. It is noteworthy that when choosing the transition, the logical condition has priority over the probabilistic one. This means that before making the probabilistic choice, one first has to position oneself in relation to the probabilistic group. As stated before, groups of transitions with the same *NS* and *CS* states but with different logical transition conditions form different probabilistic groups (the absence of logical transition conditions also distinguishes groups of such transitions from groups with logical conditions). This table (*SrcProb*) is mapped into the hacker's database (*Hacker.mdb*). At each initialization of the component, all the recalculated probabilities are reset to the initial state. This is done by introducing additional fields in the table, in which the current value of probability for each intention and different variants of transitions is stored. Thus, at the initialization of the component all the current probabilities are replaced with the corresponding values of initial probabilities that do not change in the process of the prototype operation.

6. *Scripts*. Here, the full texts of scripts specified in the section Script Name in the transition parameters table are represented. As noted above, scripts are codes written in the MAS DK Script language compilable and executable in the process of the prototype operation. Scripts are represented in a table template. One template can describe one state and include: *Entry action*; *Do action; Exit action*; *Transitions*.

The component Transitions has the following fields:

- *condition* (the condition for transition from the current state to the next state). Do not confuse with *Cond* from transition conditions. In this case, the condition indicates the instance of transition from state machine's current state to the next state. Thus, rigid behavior of each separate state machine in terms of MAS DK environment is specified;

- *next state*;
- *action*.

In all the templates of this state machine description scheme, there are no scripts describing actions for transitions, because for the description in the MAS DK environment, they were not necessary. The scripts describing *Exit actions* are used only in communicational state machines in states, where the response information from the Network Agent is received. If the last message received from the network contains a dialog completion marker, then the state machine finishes its work.

The state machine descriptions contain descriptions of scripts that are not directly associated to any state but are simply called from other scripts. Because of the congruence of the character of actions performed, such common scripts are positioned in the template in the section *Do action* or *Exit action*.

2.4.3. Component of the attack task specification

This component realizes a set of user interfaces for attack task specification by user (see paragraph 1.3.1).

The main elements of attack specification are as follows:

- ***Intention*** – malefactor's intentions to realize the attack scenario;
- ***Hacker Configuration*** – configuration of the Hacker Agent;
- ***Known Information about attacked Networks***, including its address;
- ***Object Of Attack*** – attack object defining the target of attack in more detail.

For the component's initialization, it is necessary to choose the element "Hacker Agent" in the window of the portal "*AILab Agent Library :: Portal Component*" after the loading of the Hacker Agent and the Network Agent, and there to press the button ✓. This action launches the user interface of that agent.

Fig.2.4.2 shows the main window of the user interface of the attack task specification component, which is displayed automatically after the Hacker Agent's initialization.

This window is initialized from a pre-determined script *Start_interface* of the state machines model of the Hacker Agent. This script serves for the initialization of the user interface of the applied problem during run-time.

Let us describe the main elements of the main use interface window of the attack task specification component.

1. **Intention**.

The intention is the high-level goal of the attack. The prototype uses 12 intentions in total, which could be conditionally broken into 2 classes: (1) reconnaissance and (2) implantation and threat realization.

The attacker's intentions are described in more detail in paragraph 2.4.2, where the state machines model operation is considered.

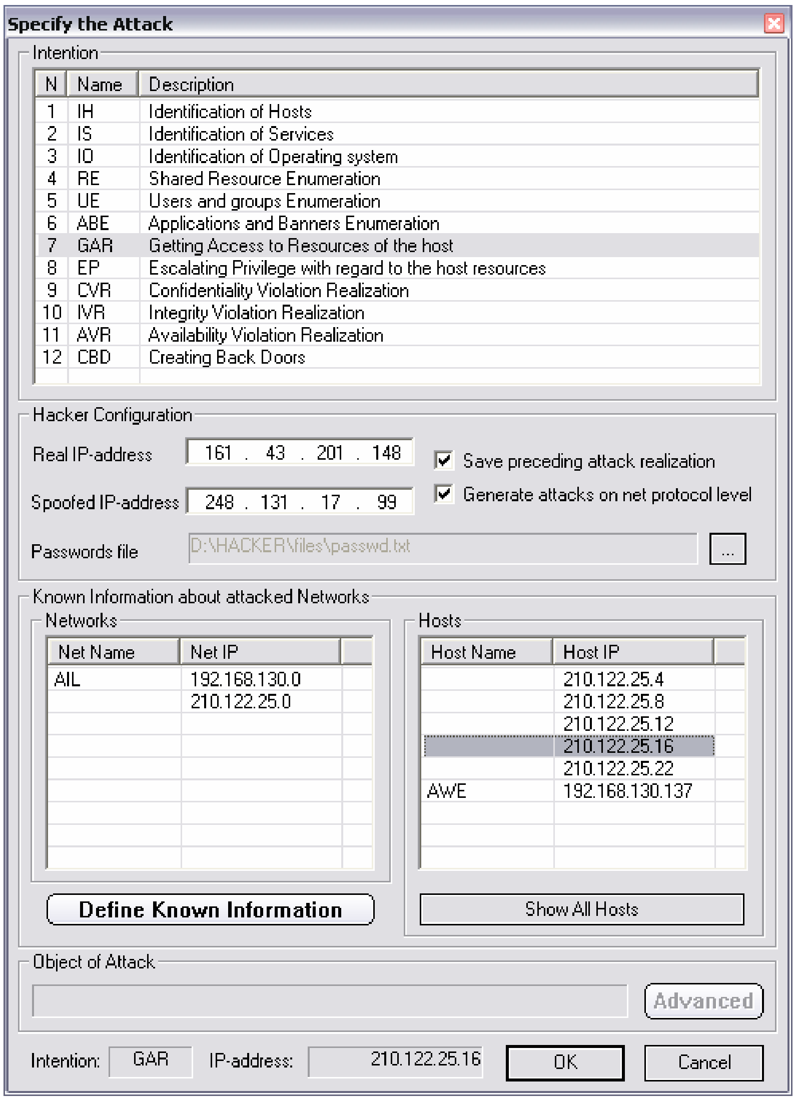
2. **Hacker Configuration**.

The settings of the Hacker Agent include the following elements:

- *Real IP-address* – Hacker Agent's real IP address. This field is mandatory. It is necessary for determining the attack scenario on both macro-level and micro-level (for forming network packets on the level of TCP/IP protocol stack);
- *Spoofed IP-address* – Hacker Agent's spoofed IP address. This field is not mandatory, unless attacks have to be generated on micro-level;
- *Password file* – path to the file with a list of words for guessing the password. This file is used only in generating attacks "Password Guessing" (PG) and "Password Cracking" (PC) on micro-level;
- *Save preceding attack realization* – the tag that determines whether the results of the previous attacks will be saved. When this parameter is initialized, the attack specified by

this component will be executed using the knowledge base formed in the previous realizations of attacks (not necessarily with the same intentions). All logs and traces are also saved in this case;

- *Generate attacks on net protocol level* – the tag that determines whether the attack will be generated on macro-level. When this parameter is initialized, besides the simulation of attack on macro-level, network packets are generated on the level of TCP/IP protocols stack. The process of attack generation on macro-level is described in more detail in paragraphs 2.4.5 and 2.6.2.



Specify the Attack

Intention

N	Name	Description
1	IH	Identification of Hosts
2	IS	Identification of Services
3	IO	Identification of Operating system
4	RE	Shared Resource Enumeration
5	UE	Users and groups Enumeration
6	ABE	Applications and Banners Enumeration
7	GAR	Getting Access to Resources of the host
8	EP	Escalating Privilege with regard to the host resources
9	CVR	Confidentiality Violation Realization
10	IVR	Integrity Violation Realization
11	AVR	Availability Violation Realization
12	CBD	Creating Back Doors

Hacker Configuration

Real IP-address 161 . 43 . 201 . 148

Spoofed IP-address 248 . 131 . 17 . 99

☑ Save preceding attack realization

☑ Generate attacks on net protocol level

Passwords file D:\HACKER\files\passwd.txt

Known Information about attacked Networks

Networks

Net Name	Net IP
AIL	192.168.130.0
	210.122.25.0

Hosts

Host Name	Host IP
	210.122.25.4
	210.122.25.8
	210.122.25.12
	210.122.25.16
	210.122.25.22
AWE	192.168.130.137

Define Known Information

Show All Hosts

Object of Attack

Advanced

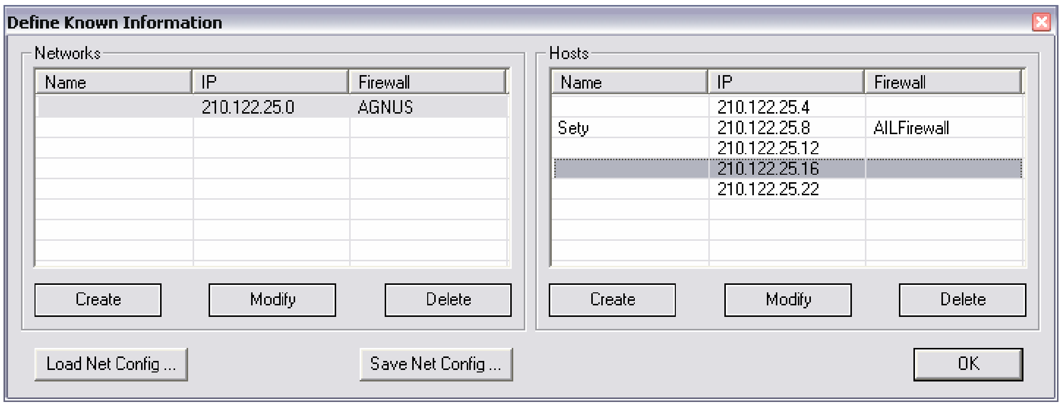
Intention: GAR IP-address: 210.122.25.16 OK Cancel

Fig.2.4.2. Example of main window of the user interface of the attack task specification

3. **Known Information about attacked Networks.**

Information specified in this section determines the Hacker Agent's knowledge base about the attacked network. The IP address of the attacked network (or host) is the attribute of this section, which in the current version of the program prototype has to necessarily be specified on the attack task specification stage. Besides the IP address, the Hacker Agent may have no additional information about the attacked host (network). In the series of attack generation experiments, this situation is only possible if the parameter "Save preceding attack realization" is turned off.

To specify additional information about the hacker's knowledge of the attacked network (hosts), the button "*Define Known Information*" should be pressed, after which a dialog window is displayed (see example in Fig.2.4.3.) In this dialog window, the user is given an opportunity to load the network configuration from the database, to save the current configuration, and modify parameters of networks or hosts by using the buttons *Create*, *Modify* and *Delete*.



Networks

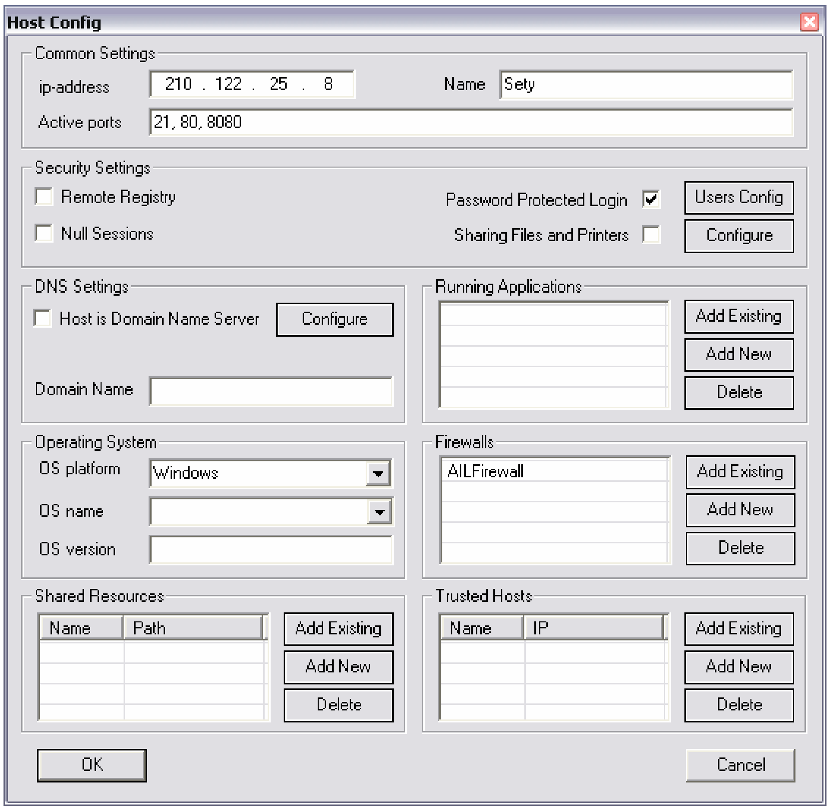
Name	IP	Firewall
	210.122.25.0	AGNUS

Hosts

Name	IP	Firewall
	210.122.25.4	
Sety	210.122.25.8	AILFirewall
	210.122.25.12	
	210.122.25.16	
	210.122.25.22	

Fig.2.4.3. Example of dialog window specifying the hacker’s knowledge of the attacked network (hosts)

Fig.2.4.4 shows an example of a dialog window initialized in modification of the information about the host known to the hacker. It is noteworthy that the program module responsible for configuring the information about the attacked network known to the hacker, is also used on the side of the Network Agent for configuring the network to be attacked, itself (see paragraph 2.5.2). Here, the sets of input data are different for the network and Hacker Agents. They specify the assumed information about the attacked network for the Hacker Agent, and the real configuration of the attacked network for the Network Agent.



ip-address: 210 . 122 . 25 . 8
Name: Sety
Active ports: 21, 80, 8080

☐ Remote Registry
☐ Null Sessions
Password Protected Login ☑
Sharing Files and Printers ☐

☐ Host is Domain Name Server
Domain Name:

OS platform: Windows
OS name:
OS version:

Firewalls: AILFirewall

Fig.2.4.4. Example of dialog window specifying the hacker’s knowledge of the attacked host

The following are the main elements of a single host configuration dialog window:

1) *Common Settings* – common settings including IP address, name of the host, as well as the list of active ports;
2) *Security Settings* – security settings for the host, including the following parameters:
   - *Remote Registry*;
   - *Null Sessions* – whether null sessions are allowed in the host registry (only for Windows platforms);
   - *Password Protected Login* – a parameter that signifies presence of a password to enter the system. By clicking on the "*User Config*" button, one can change the user settings (see paragraph 2.5.2 for more detail);
   - *Sharing Files and Printers* – network configuration parameter responsible for the possibility of sharing files and printers of this host in the local network (if the host is in a network). By using the "*Configure*" button, one can additionally configure the sharing parameters;
3) *DNS settings* – settings of DNS parameters. In this section, there should be:
   - *Host is Domain Name Server* – parameter indicating if the host is in fact, a DNS server or not. If it is, its detailed configuration is available;
   - *Domain Name* – the host's domain name (pertains to all hosts of a local network organized by the domain principle);
4) *Operating System* – parameters of the host's operating system, including platform (Windows/Unix), type (name) and version of the OS;
5) *Running Applications* – information on the applications running on host;
6) *Firewalls* – list of firewalls used that are known to the hacker. In this section, firewalls are only enumerated;
7) *Shared Resources* – list of shared resources, with resource name and path;
8) *Trusted Hosts* – trusted hosts for that specific host, with name and IP address.

The windows of the user interface used for configuring networks and hosts are represented in greater detail in paragraph 2.5.2.

One should notice that after the information about the attacked network (hosts) that is known to the Hacker Agent has been described, all information is stored in the Hacker Agent's database in the tables that are storage for instances of notions of the application domain ontology of the Hacker Agent (see paragraph 2.4.1).

4. **Object of Attack.** This element of *attack task specification* serves for specialization of the object of attack's parameters, such as files, running applications, user accounts, etc. This element is not implemented in the current version of the prototype.

After specification of data for the prototype's operation, they are recorded in the table *Objective* of the Hacker Agent's database. Here, the attack task specification component terminates its operation, and control is delegated to the Hacker Agent's state machines model.

2.4.4. Component calculating probabilities of Hacker Agent's actions

The component calculating probabilities of agent-hacker's actions intend for modeling of probabilistic (stochastic) behavior of Hacker Agent state machine model under decision-making regarding further actions.

The input data for the component's operation are entered into the table *SrcProb* of the hacker's database *MainHack*. Due to the table being too cumbersome, having about 40 fields and more than 550 records, we will describe the fields of this table below without specifying the concrete values.

Fields of the table SrcProb:

(1) ***ClassAuto*** – filed of the type *string* that contains the (abbreviated) name of the state machine, in which the probabilistic transition occurs that corresponds to one record in the table;

(2) ***SrcState*** (source state) – the current state of the state machine *ClassAuto*, from which the probabilistic transition occurs that corresponds to one record in the table;

(3) ***DstState*** (destination state) – the state of the state machine *ClassAuto*, into which the probabilistic transition occurs from the current state *SrcState*;

(4) ***Condition*** – number of the logical condition for the transition from the state *SrcState* into *DstState* of the state machine *ClassAuto*; the value *Condition* = 0 means the absence of a logical condition for the transition that corresponds to the record in the table;

(5) ***Prob_1 – Prob_12*** – fields of the type *double*, each one of which contains the probability of transition from the state *SrcState* into the state *DstState* of the state machine *ClassAuto* when the logical condition *Condition* is met for each of the 12 intentions of the attacker. If the value of the field *Prob_x*, where $x \in [1..12]$, equals 0, this means that for the intention number *x* there are no transitions from the state *SrcState* into the state *DstState* in the state machine *ClassAuto* when the *Condition* is met. On the state chart in the state machine descriptions in this case there is no connection between the states that correspond to *SrcState* and *DstState* in the table *SrcProb*;

(6) ***K_1 – K_12*** – recalculation factors for current probabilities (*xProb_1 – xProb_12*) for all transitions from the state *SrcState* of the state machine *ClassAuto* with the *Condition* being met. These transitions (from one state of the state machine into all the other states, with the *Condition* being met) form separate probabilistic groups divided by thick horizontal lines in the transitions parameters tables of the state machines descriptions. The recalculation of probabilities is performed after the transition from the state *SrcState* into the state *DstState* has been chosen and performed. The role of the factors in modeling the probabilistic behavior of the state machine model and the algorithm for probability recalculation are further considered below. The factors, as well as probabilities, may assume any values between zero and one;

(7) ***xProb_1 – xProb_12*** – current probabilities for each of the attacker's 12 intentions. At the launch of the state machine model, all *Prob_j = xProb_j*, where $j \in [1..12]$. In the process of operation the current probabilities vary, but only in regard to the actualized intention j. One can get an impression of the fields for the current probabilities being redundant. In fact, since in the duration of one run the intention initially specified in the component *TargetObjectiv* does not change, one field should be enough to store the current probabilities. The redundancy is built in on purpose, in order to provide the possibility of restarting the prototype with the parameters obtained at the previous stage. Thus, at the restart, the intentions can be either repeated from the previous start, or made new. A variant with multiple intentions specification may be realized.

Before we describe the component, let us define the notion of "probabilistic group" used by the authors of this project. A probabilistic group is a group of transitions from the source state of the state machine to all the other destination states, and the set of destination states may include the source state, under the condition that all transitions are only possible if the certain *Condition* is fulfilled.

The sums of probabilities of all such transitions for each of the 12 intentions equal 1 or 0. The zero sum implies that for this intention, there are no transitions from the source state into the destination states represented in the probabilistic group. This also implies that for this intention, there are also no transitions into the source state from any other states. Thus, a rigid behavior of the state machine model is specified on the database level.

The following two basic algorithms have been realized in the component:

1) the algorithm for choosing the transition from the source state *Throwing_dice*;

2) the algorithm for recalculating (normalizing) the probabilities within a probabilistic group *Normalize_prob*.

Transitions between states are deterministic and unique. The uniqueness of a transition is determined by the following rule: "there are no two transitions within the same state machine from the same source state into the same destination state with the same number of the logical condition of transition".

Let us consider the possible variants of relationship between two different transitions A and B within the same state machine:

SrcState (A) = *SrcState* (B), *DstState* (A) = *DstState* (B), *Condition* (A) ~= *Condition* (B) – the transitions belong to different probabilistic groups;

SrcState (A) = *SrcState* (B), *DstState* (A) ~= *DstState* (B), *Condition* (A) ~= *Condition* (B) – the transitions belong to different probabilistic groups;

SrcState (A) = *SrcState* (B), *DstState* (A) ~= *DstState* (B), *Condition* (A) = *Condition* (B) – the transitions belong to the same probabilistic group;

SrcState (A) ~= *SrcState* (B) – the transitions belong to different probabilistic groups with any values of *DstState* and *Condition*.

Thus, the uniqueness of the transition within a probabilistic group can be determined by the name of the destination state.

The attributes *m_Prob_limit_2* and *m_Prob_limit_1* for the current transition (2) and the previous transition (1) are sums of all probabilities of transitions including the probability of the transition itself (2 – for the current one, 1 – for the previous one).

Let us clarify this by an example. Let us assume there are 3 transitions (1, 2, and 3) that form a probabilistic group with the corresponding probabilities of 0.1, 0.3, and 0.6. For the transition 1, attributes *m_Prob_limit_2* and *m_Prob_limit_1* will equal 0 and 0.1 accordingly, for the transition 2 – 0.1 and 0.4, for the transition 3 – 0.4 and 0.6. These probabilistic limits are used in the first algorithm for the proportional "throwing of the dice" (choosing the range through the random number method between 0 and 1).

A probabilistic group may consist of one transition. In this case, two variants are possible for the probabilities and, accordingly, for the values of probabilistic limits for such transition:

- either all three attributes equal 0 (in this case, there is no such transition for this intention),
- or all three attributes equal 1 (the transition is probabilistically unconditional).

Attributes *m_Obj, m_SM, m_xstate* and *m_condition* correspond to the number of intention, the abbreviated name of the state machine, the source state, and the number of the logical condition for the transition (see a description of the notion *Step* of the application domain ontology). They are needed for choosing the necessary probability from the table and the corresponding recalculation factor.

It is noteworthy to describe the functions *Throwing_dice()* and *Normalize_prob()*, which realize the mechanism of the probabilistic choice and the normalization of transitions probabilities within a probabilistic group, need to be considered in more detail.

The function *Throwing_dice* operates probabilistic limits *m_Prob_limit_1* and *m_Prob_limit_2* for each of the transitions that form the probabilistic group. First, a random fractional number *x* between 0 and 1 is generated. Then, for each of the transitions, the condition for this number fitting within the range [m_Prob_limit_1 .. *m_Prob_limit_2*] is checked. As soon as this condition is met for one of the transitions from the probabilistic group, this transition is chosen.

Thus, for the example of three transitions considered above, in case the random number equals 0.3958, the transition with the lower and higher limits of 0.1 and 0.4 will be chosen, i.e. transition 2.

Probabilities reduction (normalization) function *Normalize_prob(CString Trans_Name)* is called every time after the transition has been chosen. The normalization consists in multiplying the current probability *xProb_j* of the chosen transition (for convenience, the variables here are represented by the corresponding filed/record values in the table) by the corresponding factor *K_j*.

Later, all probabilities of the intention *j* for all transitions of the probabilistic group, including the new probability of the chosen transition, are summed up.

After that, all probabilities of the intention *j* for all the summarized transitions are divided by that sum. The resultant probabilities are updated in the table *SrcProb*. The resultant probabilities after this procedure equal 1 in sum, and consequently, form a probabilistic group. Thus, we can consider the probabilities of transitions for this probabilistic group normalized by the factor *K_j* of the chosen transition for the intention *j*.

Let us consider an example of the normalization mechanism at work. Let us suppose there are the following values of probabilities and factors:

	1	2	3
P	0.1	0.3	0.6
K	0.5	0.4	0.5

Let us assume that transition 2 was chosen at the previous step. Let us calculate the new values:

1) $P_2=0.3*0.4 = 0.12$.

2) $\sum P_j = 0.1+0.12+0.6 = 0.82,\ j\in 1..3$.

3) $P_1=0.1/0.82=0.1220$; $P_2=0.12/0.82=0.1463$; $P_3=0.7317$.

Thus, $P_1+P_2+P_3 = 1$, which was to be proved.

If the factor K=1, then the current probability of the chosen transition does not change, and consequently, the current probabilities of the rest of the transitions from the probabilistic group do not change.

We have not clarified so far what is the purpose of probability recalculation factors within the probabilistic groups. The thing is that in the course of the "unfolding" of an attack scenario, a state machine model may be in the same state more than once.

This corresponds to the actions of a hacker who can use the same exploit several times in his attack. Here, he either succeeds in realizing his intention (the exploit is effective) or discontinues trying to realize the attack through using this exploit because he understands that it is not likely to yield any results in this session. The exploit in question can be inappropriate for the specific scenario, it can contain program errors, it can seldom work, etc. Depending on the circumstances, the hacker may use this exploit very seldom or very often.

The probability of the hacker making the decision to stop using the exploit completely diminishes with each new usage. At the same time, the probability of the hacker using another exploit from the probabilistic group or abandoning these exploits increases proportionally. The diminishing (increase) of probabilities is achieved through using the factor *K*.

2.4.5. Network traffic generator

This component is designed to realize the lower ("physical") level of attack generation within the attack simulator prototype.

In the current version of the prototype, the network traffic generation is only implemented for certain network attacks. Those attacks are picked so that they represent different classes of attacks and (or) malefactors' intentions specified in the application domain ontology. The authors have not tasked themselves with implementing all attack actions on lower level. The main emphasis has been made on developing the general approach to generating the network traffic in using the attack simulator prototype and assessing its feasibility and effectiveness.

To assess the prototype's effectiveness, *three different classes of attack* have been chosen, for which network packet generation was realized:

1) *Port scanning*, including subclasses "Port Scanning" (SPIH) and "Port Scanning during Identification of Services" (SPIS). On the lower level of attack class "Port Scanning" (SPIH), attacks "`TCP connect scan`" (`STIH`) and "`TCP SYN scan`" (`SSIH`) were implemented. On the lower level of attack class "Port Scanning during Identification of Services" (SPIS), attacks `TCP connect scan" (ST), "TCP SYN scan" (SS), "TCP FIN scan" (SFI), "TCP Xmas Tree scan" (SX), "TCP Null scan" (SN), "UDP scan" (SU), "Half scan"` were implemented;

2) *Denial of service*, based on the implementation of SYN flood (SF) attack;

3) *Password cracking* through guessing password, based on the implementation of attacks "Password Guessing" (PG) and "Password Cracking" (PC).

Let us show *the place of attacks implemented on lower level* in the fragment of the domain ontology "Computer network attacks". *The computer network attacks domain ontology* represents knowledge specified in terms of the basic notions of the computer network attacks domain. The shift of the corresponding notion shows a dependence of a shifted notion from the situated above non-shifted or less-shifted one. Each node of the ontology is identified in the following manner:

`<Node marker>: <Node name (interpretation)> (<Node identifier>) (<Explanations (may be absent)>).`

For example, the node “Network Attack” is identified like this: `A: Network Attack (1)`, where `A` is the node marker, `Network Attack` – the name (interpretation) of the node, `(1)` – the node identifier.

The nodes of the ontology of different hierarchy levels have their own numeration, which is performed at each level in sequence, left to right. The node identifier is set by placing in parenthesis the numbers of the nodes of higher levels connected to this node and the number of this node separated by space. These numbers are specified in the node identifier in sequence top to bottom in accordance with the hierarchy of nodes of the ontology. E.g., a node specified “`SPIH: Port Scanning (1 1 1 2)`” has the identifier `(1 1 1 2)`. This means that this node is connected top to bottom to the nodes “`A: Network Attack (1)`”, “`R: Reconnaissance (1 1)`”, and “`IH: Identification of Hosts (1 1 1)`”, and also possesses number 2 at its own level of the hierarchy.

The location of attacks “`TCP connect scan” (STIH)` and “`TCP SYN scan” (SSIH)` of attack class *“Port Scanning” (SPIH)* is as follows:

```
A: Network Attack (1)
  Part of
   •  R: Reconnaissance (1 1)
      Part of
       •  IH: Identification of Hosts (1 1 1)
          Kind of
           •  SPIH: Port Scanning (1 1 1 2)
              (different hosts are scanned before detection of one “listening” port)
              Kind of
               •  STIH: TCP connect scan (1 1 1 2 1)
               •  SSIH: TCP SYN scan (1 1 1 2 2)
               •  ...
```

The location of attacks `“TCP connect scan” (ST), “TCP SYN scan” (SS), “TCP FIN scan” (SFI), “TCP Xmas Tree scan” (SX), “TCP Null scan” (SN), “UDP scan” (SU), “Half scan”` of attack class *“Port Scanning during Identification of Services” (SPIS)* is as follows:

```
A: Network Attack (1)
  Part of
   •  R: Reconnaissance (1 1)
      Part of
       •  IS: Identification of Services (1 1 2)
          Kind of
           •  SPIS: Port Scanning during Identification of Services (1 1 2 1)
              (as a rule, all significant for attacking ports are scanned)
              Kind of
               •  ST: TCP connect scan (1 1 2 1 1)
               •  SS: TCP SYN scan (1 1 2 1 2)
               •  SFI: TCP FIN scan (1 1 2 1 3)
               •  SX: TCP Xmas Tree scan (1 1 2 1 4)
               •  SN: TCP Null scan (1 1 2 1 5)
               •  SU: UDP scan (1 1 2 1 6)
               •  HS: Half scan (1 1 2 1 7)
               •  ...
```

The attack `SYN flood (SF)` is located in the ontology in two places:

```
A: Network Attack (1)
  Part of
   •  I: Implantation and threat realization (1 2)
      Part of
```

```
    •  GAR: Getting Access to Resources (1 2 1)
      - For Unix / Linux
      Kind of
        •  CSS: Combined IP spoofing on SunOS v.1.4.x (1 2 1 12)
          Seq of
            o   DS: Denial of Service (DoS) attack (on trusted host) (1 2 1 12 1)
              Kind of
                •  SF: SYN flood (storm of inquiries on installation of TCP-
                     connections) (1 2 1 12 1 1)
```

```
A: Network Attack (1)
  Part of
  •  I: Implantation and threat realization (1 2)
    Part of
    •  TR: Threat Realization (1 2 4)
      Kind of
      •  AVR: Availability Violation Realization (1 2 4 3)
        Kind of
        •  DS: Denial of Service (DoS) attack (1 2 4 3 1)
          Kind of
          •  SF: SYN flood (storm of inquiries on installation of TCP-
               connections) (1 2 4 3 1 1)
```

The location of attacks “Password Guessing” (PG) and “Password Cracking” (PC) is as follows:

```
A: Network Attack (1)
  Part of
  •  I: Implantation and threat realization (1 2)
    Part of
    •  GAR: Getting Access to Resources (1 2 1)
      - For Windows 9X, ME
      Kind of
      •  DCSR: Direct Connection to a Shared Recourse (1 2 1 1)
        Kind of
        •  UFPS: Use of File and Print Sharing (1 2 1 1 1)
          Kind of
          •  PG: Password Guessing (for example, with BF tool of Legion) and
             access realization (1 2 1 1 1 2)
```

```
A: Network Attack (1)
  Part of
  •  I: Implantation and threat realization (1 2)
    Part of
    •  EP: Escalating Privilege (1 2 2)
      Kind of
      •  PC: Password Cracking (1 2 2 1)
```

Part of the necessary parameters for generating the network traffic is set at the attack task specification stage. Those are the following parameters:

- hacker’s IP address;
- spoofed IP address;
- password file (for guessing password).

However, this is a very incomplete list of parameters. Such parameters as the port from which an attack is realized, scanned ports range, timeouts, etc. are specified rigidly in the scripts of the state machine model, from which the functions, which generate the corresponding traffic, are initialized.

The low-level attacks described above are implemented through three executable modules. The library *libnet-1.0.2f* [Libnet] was used to implement these modules.

Let us consider the mechanism of parameterization of low-level attacks, as well as the order of their initialization from the corresponding scripts.

For each of the executable modules, three global exportable functions are provided in the kernel component of the Hacker Agent. These are the functions *_ScanPorts*, *_SYNflood* and *_FTPcrack*.

Those functions are initialized from the scripts, in which the execution of the corresponding attacks is stipulated. Parameters of all three functions are the input parameters, the result of the functions' operation is not checked, for it is not necessary – the traffic generated by he Hacker Agent is not processed on the Network Agent side.

Let us describe the parameters of these functions.

1. The parameters of the function *_ScanPorts(sType, sNetI, Objective_OwnIP, sHackerPort, Objective_Host, sHostPorts, sTimeOut)*:

- *sType* – scanning type;
- *sNetI* – number of the network interface, which will be listened to in order to receive a response from the scanned host. To obtain the numbers, one must use *Windump –D* . Usually equals 2;
- *Objective_OwnIP* – IP address and port of the host – source of packets, not used in TCP connect scan;
- *sHackersPort* – hacker's port, from which the attack is executed;
- *Objective_Host* – IP address of the scanned host;
- *sHackerPort* – string of scanned ports enumeration (ports are separated by commas, dash "-" denotes range of ports), for example: "1-23,80,143-1024";
- *sTimeOuts* – scanned host response waiting time (1 second by default).

2. The parameters of the function *_SYNflood (Objective_ShamIP, Objective_Host, sHostPort)*:

- *Objective_ShamIP* – spoofed IP address of the paskets' "source";
- *Objective_Host* – IP address of the attacked host;
- *sHostPort* – port of the attacked host, to which the packets are sent.

3. The parameters of the function *_FTPcrack (Objective_Host, Objective_PSWfile, User_ID)*:

- *Objective_Host* – IP address of the attacked ftp-server;
- *Objective_PSWfile* – password dictionary file;
- *User_ID* – name of one of the users of the attacked host, known to the hacker.

For each of the attack classes described let us consider examples of scripts used for initialization of functions *_ScanPorts*, *_SYNflood* and *_FTPcrack* of network traffic generation (Tab.2.4.2, Tab.2.4.3 and Tab.2.4.4).

Tab.2.4.1. Script of agent's behavior in state SSIH of the state machine SPIH

Entry	
Entry action	
State action	
Do action	Step.xState="SSIH"; Step.Condition=0;
	IF (Objective.LowLevel=1) THEN
SPIH_SSIH_Do	str="sS"; str1="2"; str2="1050"; str3="1-23,80,1000-1024"; str4="1";
	IF (Objective.Net!=1) THEN
	_ScanPorts (str, str1, Objective.OwnIP, str2, Objective.Host, str3, str4); ENDIF;
	IF (Objective.Net=1) THEN
	IF (KnownLANs.Exist(KnownLANs.IP=Objective.Host)) THEN
	REPEAT
	IF (KnownLANs.Exist(KnownLANs.HostIP!="") AND
	(KnownLANs.IP=Objective.Host)) THEN
	_ScanPorts (str, str1, Objective.OwnIP, str2, KnownLANs.HostIP, str3, str4);
	ENDIF;
	UNTIL (KnownLANs.Next());
	ENDIF;
	ENDIF;
	ENDIF;
	CALLSCRIPT (Do_script);
Transitions. Condition / Next state / Action	
Exit action	

Tab.2.4.2. Script of agent’s behavior in state SF of the state machine DS

Entry				
Entry action				
State action				

Do action
 	
DS_SF_Do
	Step.xState="SF"; Step.Condition=0;
	IF (Objective.LowLevel=1) THEN
	    str="21";
	    IF (Objective.Net!=1) THEN
	        _SYNflood (Objective.ShamIP, Objective.Host, str); ENDIF;
	    IF (Objective.Net=1) THEN
	        IF (KnownLANs.Exist(KnownLANs.IP=Objective.Host)) THEN
	            REPEAT
	              IF (KnownLANs.Exist(KnownLANs.HostIP!="") AND
	                    (KnownLANs.IP=Objective.Host)) THEN
	                      _SYNflood (Objective.ShamIP, KnownLANs.HostIP, str); ENDIF;
	            UNTIL (KnownLANs.Next());
	        ENDIF;
	    ENDIF;
	ENDIF;
	CALLSCRIPT (Do_script);			

Transitions. Condition / Next state / Action				
Exit action				

Tab.2.4.3. Script of agent’s behavior in state PC of the state machine EP

Entry				
Entry action				
State action				

Do action
 	
EP_PC_Do
	Step.xState="PC"; Step.Condition=0;
	IF (Objective.LowLevel=1) THEN
	    IF (Objective.Net!=1) THEN
	      IF (User.Exist(User.ID!=“”) THEN
	        REPEAT
	            _FTPcrack (Objective.Host, Objective.PSWfile, User.ID);
	        UNTIL (User.Next());
	    ENDIF;
	    IF (Objective.Net=1) THEN
	        IF (KnownLANs.Exist(KnownLANs.IP=Objective.Host)) THEN
	            REPEAT
	              IF (KnownLANs.Exist(KnownLANs.HostIP!="") AND
	                    (KnownLANs.HostIP=Objective.Host)) THEN
	                    IF ((User.Exist(User.IP= KnownLANs.HostIP)) AND (User.ID!=“”)) THEN
	                      REPEAT
	                          _FTPcrack (KnownLANs.HostIP, Objective.PSWfile, User.ID);
	                      UNTIL (User.Next());
	                    ENDIF;
	              ENDIF;
	            UNTIL (KnownLANs.Next());
	        ENDIF;
	    ENDIF;
	ENDIF;
	CALLSCRIPT (Do_script);			

Transitions. Condition / Next state / Action				
Exit action				

Examples of parameters of network traffic generation programs, used to implement various attacks, are shown in Tab.2.4.4. Examples of the source codes of these network traffic generation programs are represented in Appendix 3. Logs of attack traces produced by calling these programs are fixed in Appendix A4.2.

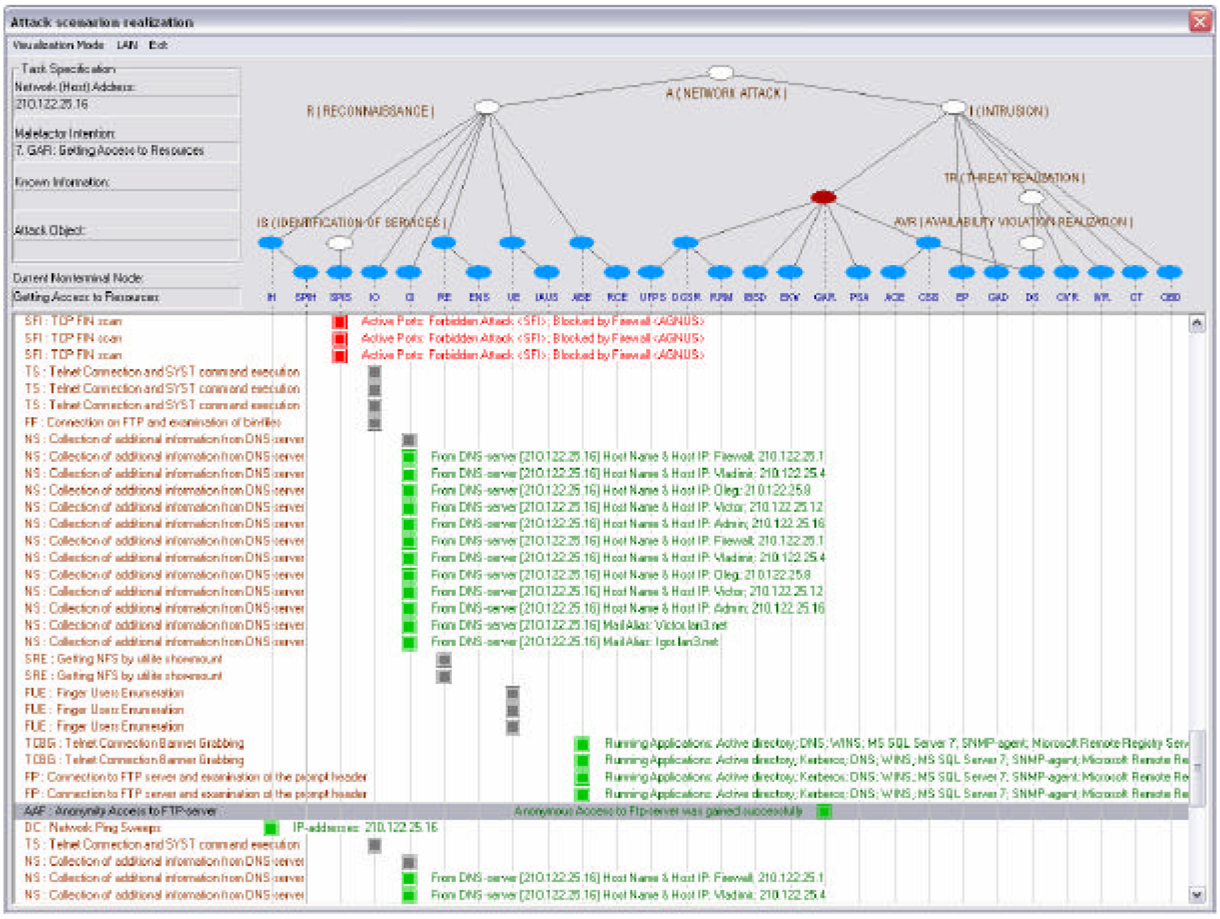
Tab.2.4.4. Examples of parameters of network traffic generation programs

Attack name	Interpretation	String of program calls
HS	Half scan	Scanports.exe –sS –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SFI	TCP FIN scan	Scanports.exe –sF –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SN	TCP Null scan	Scanports.exe –sN –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SS	TCP SYN scan	Scanports.exe –sS –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SSIH	TCP SYN scan	Scanports.exe –sS –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
ST	TCP connect scan	Scanports.exe –sT –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
STIH	TCP connect scan	Scanports.exe –sT –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SU	UDP scan	
SX	TCP Xmas Tree scan	Scanports.exe –sX –i2 –h10.0.0.21.1050 –d10.0.0.12 –p"1-1024" –t3
SF	SYN flood	SYNflood.exe –s 11.0.0.21 –d 10.0.0.12.21
PG	Password Guessing	ftpcrack.exe –d 10.0.0.12.21 –f passwd.txt–u username
PC	Password Cracking	ftpcrack.exe –d 10.0.0.12.21 –f passwd.txt–u username

2.4.6. Visualization component of the attack scenario development

This component is used for the visualization of the attack generation process. The component allows for graphic, "real-time" visualization of the "unfolding" of attack scenario.

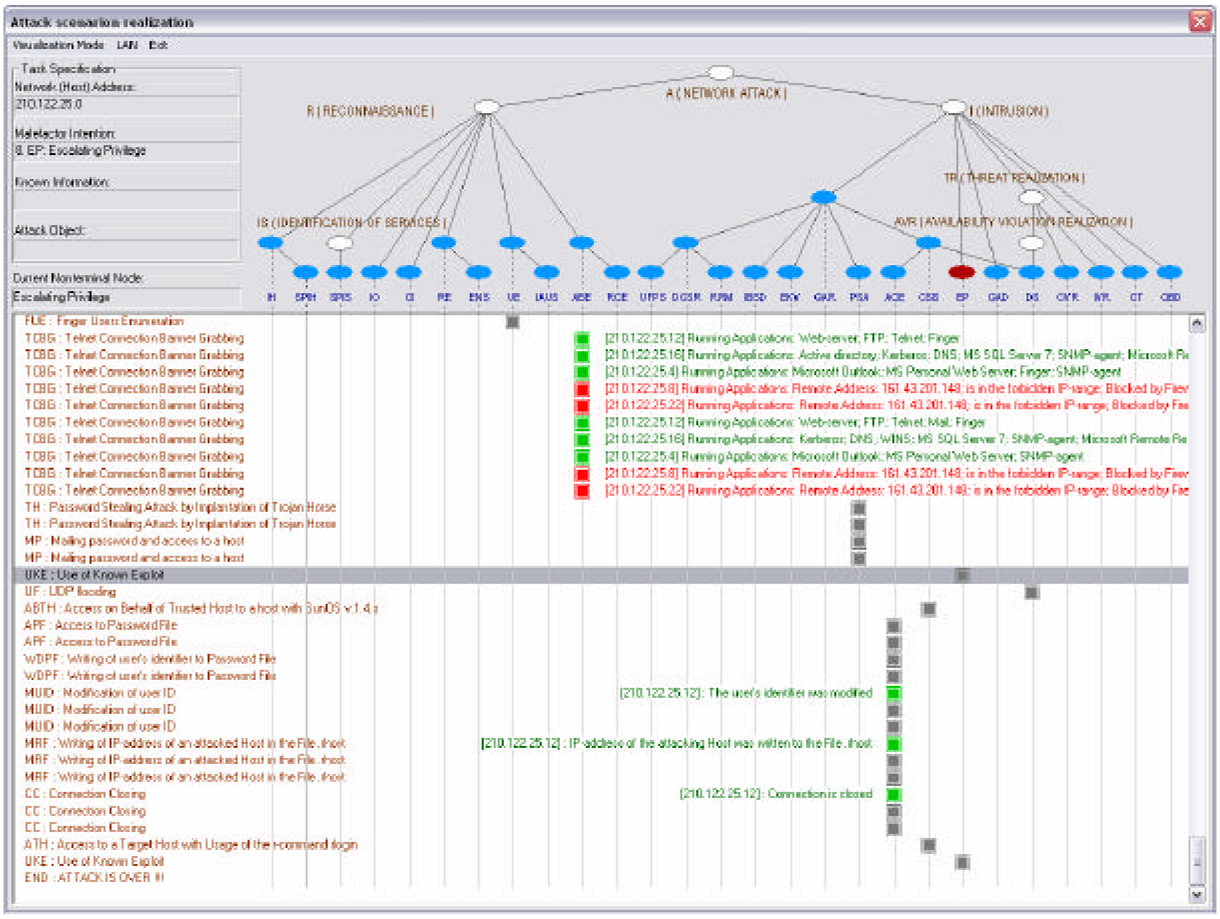
Since there are some differences in implementation between attacks aimed at a network (all hosts of a network) and at separate hosts, those attacks' visualization logs are also different. Therefore, below we will consider both variants of logs.



Task Specification
Network (Host) Address: 210.122.25.16
Malefactor Intention: 7. GAR: Getting Access to Resources
Known Information:
Attack Object:
Current Nonterminal Node: Getting Access to Resources

```
SFI : TCP FIN scan                                        Active Ports: Forbidden Attack <SFI>; Blocked by Firewall <AGNUS>
SFI : TCP FIN scan                                        Active Ports: Forbidden Attack <SFI>; Blocked by Firewall <AGNUS>
SFI : TCP FIN scan                                        Active Ports: Forbidden Attack <SFI>; Blocked by Firewall <AGNUS>
TS : Telnet Connection and SYST command execution
TS : Telnet Connection and SYST command execution
TS : Telnet Connection and SYST command execution
FF : Connection on FTP and examination of binfiles
NS : Collection of additional information from DNS-server
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Firewall; 210.122.25.1
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Vladimir; 210.122.25.4
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Oleg; 210.122.25.8
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Victor; 210.122.25.12
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Admin; 210.122.25.16
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Firewall; 210.122.25.1
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Vladimir; 210.122.25.4
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Oleg; 210.122.25.8
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Victor; 210.122.25.12
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Admin; 210.122.25.16
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Mail Alias: Victor.lan3.net
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Mail Alias: Igor.lan3.net
SRE : Getting NFS by utilite showmount
SRE : Getting NFS by utilite showmount
FUE : Finger Users Enumeration
FUE : Finger Users Enumeration
FUE : Finger Users Enumeration
TCBG : Telnet Connection Banner Grabbing                   Running Applications: Active directory; DNS; WINS; MS SQL Server 7; SNMP-agent; Microsoft Remote Registry Serv
TCBG : Telnet Connection Banner Grabbing                   Running Applications: Active directory; Kerberos; DNS; WINS; MS SQL Server 7; SNMP-agent; Microsoft Remote Re
FP : Connection to FTP server and examination of the prompt header   Running Applications: Active directory; Kerberos; DNS; WINS; MS SQL Server 7; SNMP-agent; Microsoft Remote Re
FP : Connection to FTP server and examination of the prompt header   Running Applications: Active directory; Kerberos; DNS; WINS; MS SQL Server 7; SNMP-agent; Microsoft Remote Re
AAF : Anonymity Access to FTP-server                       Anonymous Access to Ftp-server was gained successfully
DC : Network Ping Sweeps                                   IP-addresses: 210.122.25.16
TS : Telnet Connection and SYST command execution
NS : Collection of additional information from DNS-server
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Firewall; 210.122.25.1
NS : Collection of additional information from DNS-server   From DNS-server [210.122.25.16] Host Name & Host IP: Vladimir; 210.122.25.4
```

Fig.2.4.5. Example of visualization window of a single host attack scenario



```
Task Specification
Network (Host) Address:
210.122.25.0
Malefactor Intention:
8. EP: Escalating Privilege
Known Information:

Attack Object:

Current Nonterminal Node:
Escalating Privilege
```

```
FUE : Finger Users Enumeration
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.12] Running Applications: Web-server; FTP; Telnet; Finger
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.16] Running Applications: Active directory; Kerberos; DNS; MS SQL Server 7; SNMP-agent; Microsoft Re
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.4] Running Applications: Microsoft Outlook; MS Personal Web Server; Finger; SNMP-agent
TCBG : Telnet Connection Banner Grabbing   🟥 [210.122.25.8] Running Applications: Remote Address: 161.43.201.148; is in the forbidden IP-range; Blocked by Fire
TCBG : Telnet Connection Banner Grabbing   🟥 [210.122.25.22] Running Applications: Remote Address: 161.43.201.148; is in the forbidden IP-range; Blocked by Fire
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.12] Running Applications: Web-server; FTP; Telnet; Mail; Finger
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.16] Running Applications: Kerberos; DNS ; WINS; MS SQL Server 7; SNMP-agent; Microsoft Remote Re
TCBG : Telnet Connection Banner Grabbing   🟩 [210.122.25.4] Running Applications: Microsoft Outlook; MS Personal Web Server; SNMP-agent
TCBG : Telnet Connection Banner Grabbing   🟥 [210.122.25.8] Running Applications: Remote Address: 161.43.201.148; is in the forbidden IP-range; Blocked by Fire
TCBG : Telnet Connection Banner Grabbing   🟥 [210.122.25.22] Running Applications: Remote Address: 161.43.201.148; is in the forbidden IP-range; Blocked by Fire
TH : Password Stealing Attack by Implantation of Trojan Horse
TH : Password Stealing Attack by Implantation of Trojan Horse
MP : Mailing password and access to a host
MP : Mailing password and access to a host
UKE : Use of Known Exploit
UF : UDP flooding
ABTH : Access on Behalf of Trusted Host to a host with SunOS v.1.4.x
APF : Access to Password File
APF : Access to Password File
WDPF : Writing of user's identifier to Password File
WDPF : Writing of user's identifier to Password File
MUID : Modification of user ID             [210.122.25.12]: The user's identifier was modified 🟩
MUID : Modification of user ID
MUID : Modification of user ID
MRF : Writing of IP-address of an attacked Host in the File .rhost   [210.122.25.12] : IP-address of the attacking Host was written to the File .rhost 🟩
MRF : Writing of IP-address of an attacked Host in the File .rhost
MRF : Writing of IP-address of an attacked Host in the File .rhost
CC : Connection Closing                    [210.122.25.12]: Connection is closed 🟩
CC : Connection Closing
CC : Connection Closing
ATH : Access to a Target Host with Usage of the r-command rlogin
UKE : Use of Known Exploit
END : ATTACK IS OVER !!!
```

Fig.2.4.6. Example of visualization window of a network attack scenario

This section does not deal with aspects of visualization of the network and hosts initial data specification process; nor does it deal with the process of specifying the information about the network (host) to be attacked that is known to the hacker. It contains only the description of attack visualization windows.

The processes of macro-level attack generation experiments are considered in detail in paragraph 2.6.1, the network configuration processes – in paragraph 2.5.2, and the processes of entering the input data about the attacked network into the Hacker Agent's knowledge base – in paragraph 2.4.3.

The first example of the main demonstration window showing the development of attack on single host is represented in Fig.2.4.5.

It depicts the fragment of attack development for the intention 7 ("Getting Access to Resources (GAR)"), where the hacker's IP-address is 161.43.201.148 and the host IP-address is 210.122.25.16.

The second example of the main demonstration window showing the development of attack on the hole network is represented in Fig.2.4.6.

It depicts the fragment of attack development for the intention 8 ("Escalating Privileges (EP)"), where the hacker's IP-address is 161.43.201.148 and the host IP-address is 210.122.25.0.

In the indicated figures the attack information is divided on the following four groups:

(1) the *attack task specification* units are mapped in the left top of the screen;

(2) to the right of them the *attack generation tree* is visualized;

(3) the strings of *generated malefactor's actions* are placed in the left part of the screen below the attack task specification;

(4) on the right of each malefactor's action a *tag of success (failure)* and *data obtained from an attacked host (a host response)* are depicted.

The *Attack task specification section* contains the information generated by the component of the attack task specification (see paragraph 2.4.3).

The graph showing the *Attack generation tree* represents a hierarchy of the malefactor's intentions and actions of different levels which correspond to non-terminal and terminal nodes. The non-terminal high level nodes are depicted by white ellipses. The terminal nodes of the attack model correspond to blue nodes. The brown node is the node of the current step of an attack scenario execution.

The transcriptions of the blue nodes can be seen in the section "Current non-terminal node".

All non-terminal nodes are realized as state machines.

When the attack scenario is developing the strings with the following elements are appeared in the white window:

- Braun strings in left part of the diagram are descriptions of the *generated terminal malefactor's actions*.
- The result of each malefactor's action may be positive or negative. If the result is positive, the square block (designating the *tag of success*) is green, and green comments are printed from the right of the square block. The negative result means that the action was done unsuccessfully. The negative result is possible in two cases: if the attack is blocked by a firewall (in that case, the indicator and the comment are red); if the network response is negative (the indicator is grey, the comment is absent). When the string "END: Attack is over" is appeared, this means that a scenario realization is finished.

As shown in Fig.2.4.6, in the network attack implementation, each terminal action is performed on each host of the network, and in case of success or the attack being blocked by the firewall, right after the square block is the IP address of the host at which that terminal attack action was directed.

In case of success, the comment contains the decoding of the result obtained through that terminal action of the Hacker Agent, and the information obtained from the Network Agent as a result of the attacker's action (that information may be absent).

In case of the hacker's attack being blocked, the comment contains information on the reasons of the attack being blocked (either an illegal IP address of sender or receiver was detected, or the specified attack signature was detected), as well as the name of the firewall. If the attack was blocked on the level of the network firewall, then the IP address of the network is placed at the start of the comment.

After completion of the attack scenario the message "END: Attack is over" appears in the right part of the white window.

It is possible to look through the scenario tree by moving between the strings on the diagram.

The current scenario realization can be finished by closing the main dialog window. After that it is possible to begin another scenario.

2.5. Generic Network Agent

The generic Network Agent (*NetAgent*) consists of a set of component realized in MASDK, as well as a number of external components realized in MS Visual C++.

The main components of the generic agent of the class *NetAgent*:

- a fragment of the general computer network attack application domain ontology that is used by the Network Agent in its operation;
- state machines model that realizes Network Agent's behavior scenarios;
- set of scripts for controlling the Network Agent in the process of forming a response to Hacker Agent's attack actions;
- a program module that realizes the necessary functions and user interfaces created outside MASDK (including specification of computer network configuration).

2.5.1. Fragment of the ontology used by the Network Agent

Let us consider the basic notions of the computer network attack application domain ontology that are used only by the "attacked computer network" agent. Let us introduce these notions by breaking them into categories corresponding to the notions' purpose.

1. *Notions used for storing information about the network and each of its hosts*:
   1) ***LAN***;
   2) ***Host***;

3) ***TrusHosts***;
4) ***SharedRes***;
5) ***Domain***;
6) ***DomHost***;
7) ***DomLink***;
8) ***DNS1***;
9) ***DNS2***;
10) ***User***;
11) ***Security***;
12) ***Service***;
13) ***Appl***.

Except for the notion *LAN* that contains network's description (its IP address, subnet and network name), all the other notions are utilized both by the Hacker Agent and the Network Agent. The notion *KnownLANs* of Hacker Agent's fragment of ontology unlike the notion *LAN*, contains the only description of networks known to the hacker.

2. *Notions that describe the model of potential hacker in terms of the application domain (including the terms of the firewall model realization).*
   1) **Attacks**. Examples of this notion contain all the information necessary for the Network Agent to form a response to each of the known attack actions (i.e. actions represented in one of the examples of this notion). Attributes of the notion *Attacks*:
      - ***Appl*** – list of applications running on the host, for which the attack *Name* is executed. For all the attributes of this notion, a rule is used in the program realization, in accordance to which the attack *Name* is executed when only all the attributes assigned in the example of this notion, which are conditions for the attack, are fulfilled. If the attribute has not been assigned (is empty), then the condition related to that attribute is not checked. This means, for example, that if for a certain example of the notion *Attacks* the attribute *OSplatform* is not assigned, then the attack is executed on all types of OS, provided all other conditions are met;
      - ***Class*** – class, to which the attack *Name* belongs. This attribute corresponds to the attribute *Class* of the notion *Attack*;
      - ***Name*** – name of the attack (attack action). This attribute is an abbreviation and corresponds to the attribute *Name* of the notion *Attack*. This attribute is mandatory;
      - ***OSname*** – types of operating systems for which the attack *Name* is executed. Types of operating systems are separated by commas in this attribute;
      - ***OSplatform*** – OS platforms for which the attack *Name* is executed. Operating system platforms in this attribute are separated by commas (as noted above, this prototype uses two platforms, Windows and Unix);
      - ***OSversion*** – OS versions for which the attack *Name* is executed; OS versions are represented in a list and separated by commas;
      - ***Ports*** – numbers of open ports (1 through 65536) in a comma-separated list, for which the attack *Name* is executed;
      - ***Prob*** – probability of execution of the attack *Name* if all other conditions are met. This parameter is mandatory. If the attack is always executed with all the other conditions fulfilled, then the value of this attribute is 1;
      - ***Security*** – the security parameters that are not related to the firewall settings. Values of this attribute correspond to the names of the attributes of the notion *Security* (*NS, CFP, Psw, RR*);
      - ***Service*** – network services and (or) protocols (protocol stack), for which the attack *Name* is executed. The following protocols are used in the prototype: TCP, IP, UDP, Telnet, NetBIOS and ICMP;
      - ***SubClass0*** – 0th level subclass of the attack action *Name*;
      - ***SubClass1*** – 1st level subclass of the attack action *Name*;

- ***SubClass2*** – 2nd level subclass of the attack action *Name*. One of the attributes *SubClass0, SubClass1, SubClass 2* has to be assigned.

2) **Firewall.** This notion describes the firewall model. If one of the hosts of the network has a firewall with the name that corresponds to the attribute *Name* of this notion, then in the instance of the notion *Host* that corresponds to that host, or for the entire network *LAN*, *FirewallName* = *Firewall.Name*. The following are the attributes of the notion *Firewall*:
   - ***Name*** – firewall name;
   - ***AttackName*** – abbreviated name of the attack in the firewall's knowledge base;
   - ***Prob*** – probability of the execution of *AttackName*.
3) **ForbiddenLocalAddr.** Contains lists of forbidden local addresses and has the following attributes:
   - ***FirewallName*** – firewall name;
   - ***LocalIP*** – forbidden local IP address for *FirewallName*;
   - ***LocalIPRange*** – range of forbidden local addresses for the firewall *FirewallName*.
4) **ForbiddenRemoteAddr.** Contains lists of forbidden remote addresses and has the following attributes:
   - ***FirewallName*** – firewall name;
   - ***RemoteIP*** – forbidden remote IP address for *FirewallName*;
   - ***RemoteIPRange*** – range of forbidden remote addresses for the firewall *FirewallName.*

The agent "attacked computer network", upon receiving a message with the notion *Attack* from the Hacker Agent, executes the following sequence of actions:

- ✓ Checks the parameter *ip* of the incoming message (that specifies the address of the attacked network or host) to see if that host has a firewall, or if there is a network firewall (the notion *Firewall*). After that, if there is a firewall, it checks whether the firewall blocks attacks whose signatures correspond to the attribute *Name* of the incoming message, and checks the attribute *HackerIP* to see if it blocking messages from the sender address of that message. If the first condition is fulfilled, then the positive or negative response is formed according to the probability specified for that attack. The negative response sites which firewall and from which host the attack was blocked. This is done in order to ensure clarity of the attack trace for the user. If the second condition is fulfilled, the attack is definitely blocked, and a notification of deterred threat is sent. If neither condition is fulfilled (there is no firewall, or there is no attack signature, or (and) the sender's IP address is not forbidden), the agent "attacked network" tries to form a positive response, for which it executes the second action.
- ✓ Checks for availability of the attack *Name* and, if the attack is found in the Network Agent's knowledge base (set of assigned examples of the notion *Attack*), checks the conditions for the execution of that attack. If all the conditions are fulfilled, then the probability of successful response is calculated, and, if successful, the response is formed.

3. *The notions used for exchange of information between the Hacker Agent and the Network Agent through their attributes*:
   1) **Attack**. If a positive response is formed, the agent network duplicates all assigned attributes of the incoming message in its response, and adds the attributes that are presented in the model of the Network Agent's reaction.

2.5.2. Component of specification of computer network configuration

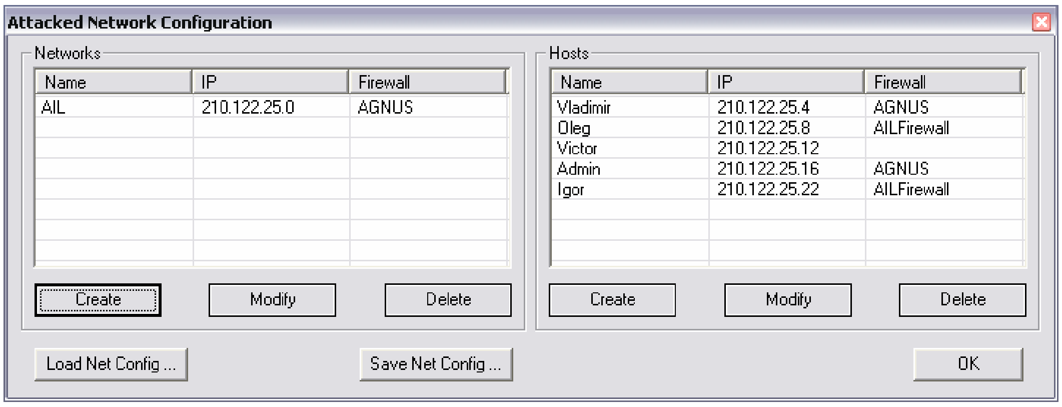
This component is used for configuration of the attacked computer network through filing the Network Agent's knowledge base.

For the component's initialization, it is necessary to choose the element "Network Agent" in the window of the portal "*AILab Agent Library :: Portal Component*" after the loading of the Hacker

Agent and the Network Agent, and there to press the button ✓. This action launches the user interface of that agent.

One should notice that the network should be configured (if necessary) before the state machine model of the Hacker Agent is started. Moreover, the user interface of the Hacker Agent may already be functioning (see paragraph 2.4.3), but the *OK* button has not been pressed. Otherwise, attack starts generating on the network which has been configured previously.

Fig.2.5.1 shows the dialog window that appears immediately after the Network Agent's user interface has been started.


Attacked Network Configuration

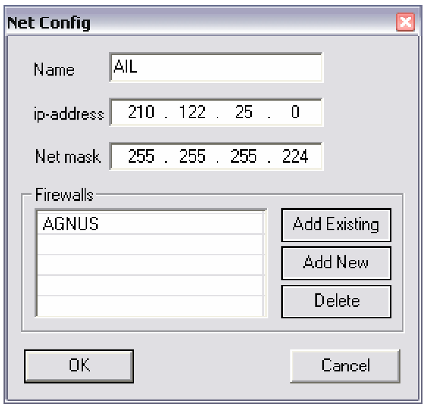
Networks

Name	IP	Firewall
AIL	210.122.25.0	AGNUS

Hosts

Name	IP	Firewall
Vladimir	210.122.25.4	AGNUS
Oleg	210.122.25.8	AILFirewall
Victor	210.122.25.12	
Admin	210.122.25.16	AGNUS
Igor	210.122.25.22	AILFirewall

Fig.2.5.1. Example of initial dialog window for configuration of the network attacked


Net Config

Name: AIL

ip-address: 210 . 122 . 25 . 0

Net mask: 255 . 255 . 255 . 224

Firewalls: AGNUS

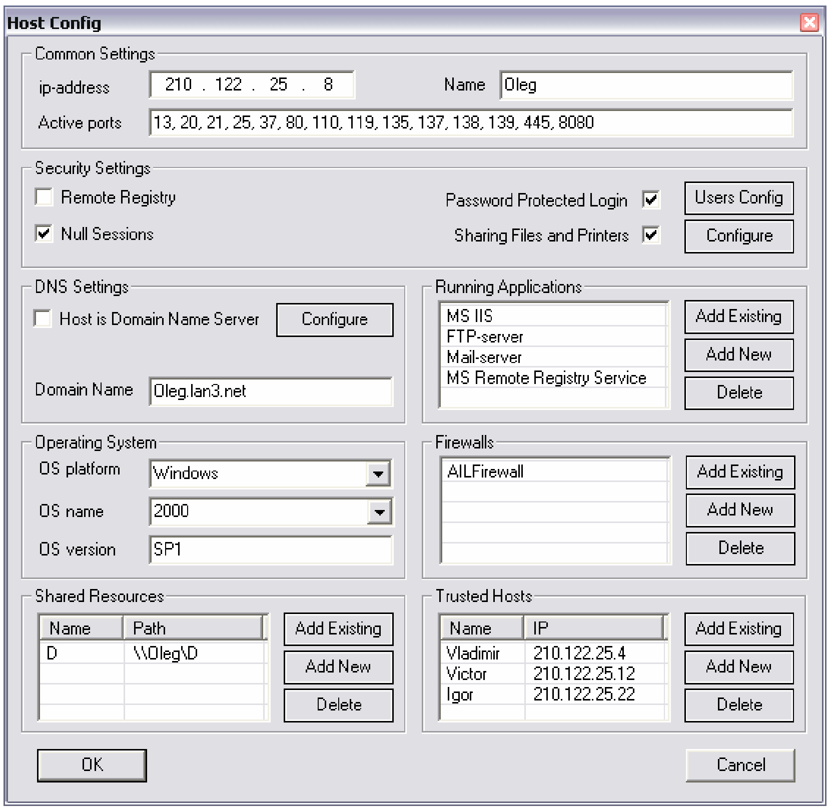
Fig.2.5.2. Example of dialog window for modification of the attacked network parameters

This dialog window is identical to the window shown in Fig.2.4.3 in the description of the Hacker Agent; however, it is used for entering the information into the Network Agent's knowledge base. In this window, the user is given an opportunity to load the available network configuration from the database, to save the current configuration, or to create or modify various parameters of networks or separate hosts though the use of buttons *Create*, *Modify* and *Delete*.

Fig.2.5.2 shows an example of a dialog window displayed during the modification of network information.

This window allows for specifying the main parameters of the computer network in general. Configuring the network in general lies in specifying the IP address, the network domain name or workgroup name, subnet mask, and the (network) firewalls used for the protection of the entire network. By using the buttons "*Add Existing*", "*Add New*" and "*Delete*", on can initialize configuring network firewalls (see Fig.2.5.6), their creation and deletion from this window.

Fig.2.5.3 shows an example if host configuration dialog window, which is started from the initial configuration window for the attacked network.


Host Config

Common Settings

ip-address	210 . 122 . 25 . 8	Name	Oleg
Active ports	13, 20, 21, 25, 37, 80, 110, 119, 135, 137, 138, 139, 445, 8080		

Security Settings

[ ] Remote Registry
[✓] Null Sessions
Password Protected Login [✓] Users Config
Sharing Files and Printers [✓] Configure

DNS Settings

[ ] Host is Domain Name Server Configure
Domain Name: Oleg.lan3.net

Running Applications

- MS IIS
- FTP-server
- Mail-server
- MS Remote Registry Service

Operating System

OS platform	Windows
OS name	2000
OS version	SP1

Firewalls

- AILFirewall

Shared Resources

Name	Path
D	\\Oleg\D

Trusted Hosts

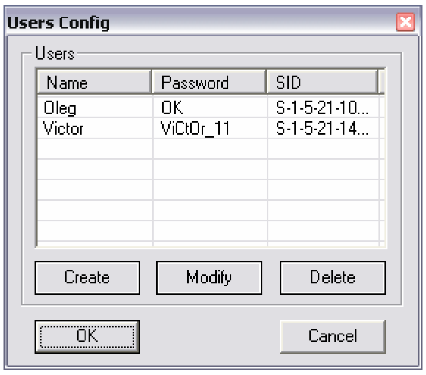
Name	IP
Vladimir	210.122.25.4
Victor	210.122.25.12
Igor	210.122.25.22

Fig.2.5.3. Example of dialog window for host parameters configuration

The following are the main elements of a single host configuration dialog window:

1) *Common Settings* – common settings including IP address, name of the host, as well as the list of active ports;
2) *Security Settings* – security settings for the host, including the following parameters:
   - *Remote Registry*;
   - *Null Sessions* – whether null sessions are allowed in the host registry (only for Windows platforms);
   - *Password Protected Login* – a parameter that signifies presence of a password to enter the system. By clicking on the "*User Config*" button, one can change the user settings (see Fig.2.5.4);
   - *Sharing Files and Printers* – network configuration parameter responsible for the possibility of sharing files and printers of this host in the local network (if the host is in a network). By using the "*Configure*" button, one can additionally configure the sharing parameters;
3) *DNS settings* – settings of DNS parameters. In this section, there should be:
   - *Host is Domain Name Server* – parameter indicating if the host is in fact, a DNS server or not. If it is, its detailed configuration is available (see Fig.2.5.5);
   - *Domain Name* – the host's domain name (pertains to all hosts of a local network organized by the domain principle);
4) *Operating System* – parameters of the host's operating system, including platform (Windows/Unix), type (name) and version of the OS;
5) *Running Applications* – information on the applications running on host;
6) *Firewalls* – list of firewalls used that are known to the hacker. In this section, firewalls are only enumerated. Issues of firewall parameters specification are considered below;
7) *Shared Resources* – list of shared resources, with resource name and path;
8) *Trusted Hosts* – trusted hosts for that specific host, with name and IP address.

Fig.2.5.4. shows an example of the configuration dialog window for host's user account.



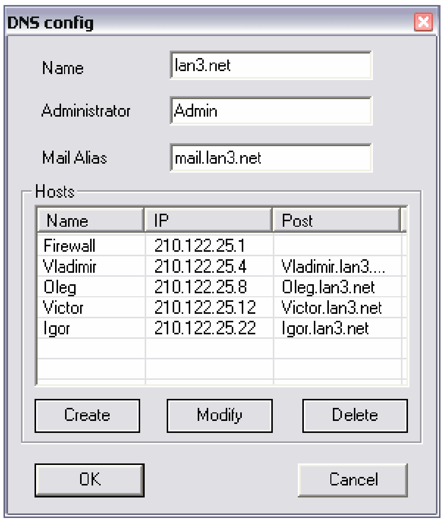
Name	Password	SID
Oleg	OK	S-1-5-21-10...
Victor	ViCtOr_11	S-1-5-21-14...

Fig.2.5.4. Example of the configuration dialog window for host’s user account

In accordance with the application domain ontology, besides the host’s IP address, a user is identified by the following three attributes:

- user name;
- password to enter the system;
- unique security ID (SID) (for Windows platform).

Fig.2.5.5 shows a configuration dialog window for a DNS server and its accounts.



Name	lan3.net
Administrator	Admin
Mail Alias	mail.lan3.net

Name	IP	Post
Firewall	210.122.25.1	
Vladimir	210.122.25.4	Vladimir.lan3....
Oleg	210.122.25.8	Oleg.lan3.net
Victor	210.122.25.12	Victor.lan3.net
Igor	210.122.25.22	Igor.lan3.net

Fig.2.5.5. Example of dialog window for DNS server parameters configuration

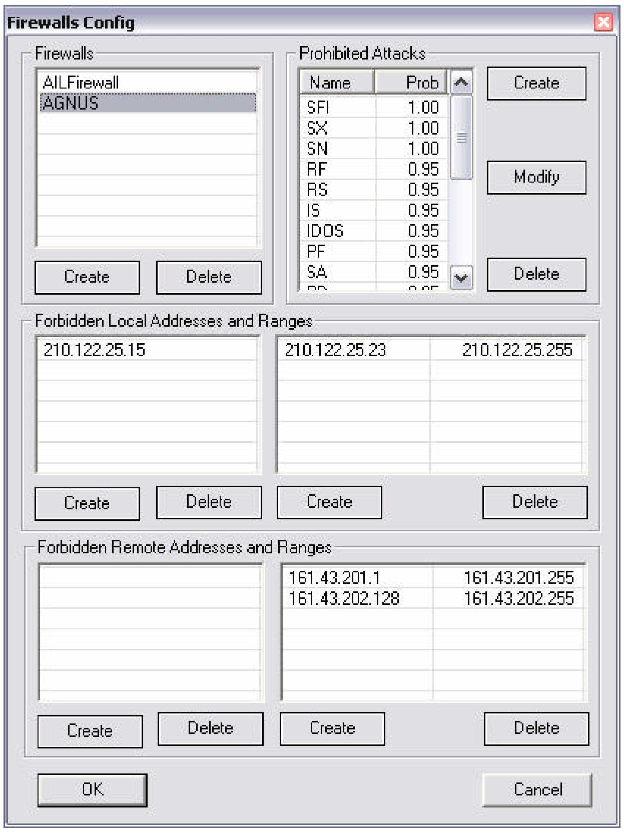
This dialog allows for modification of the main parameters of the DNS server, including the hosts’ accounts.

Fig.2.5.6 shows an example of a dialog window for firewall parameters configuration. This dialog window may be started from both the network parameters specification window and the host parameters specification window.

The firewall configuration dialog window *“Firewalls Config”* contains the following elements:

1) *Firewalls* – list of all firewalls described in the ontology fragment;
2) *Prohibited Attacks* – list of prohibited attacks whose signatures are detected with a specified probability by the specified firewall (by a firewall selected in the list *Firewalls*);
3) *Forbidden Local Addresses and Ranges* – list of local IP addresses and their ranges, outgoing connections from which are forbidden by the firewall;
4) *Forbidden Remote Addresses and Ranges* – list of remote IP addresses and their ranges, outgoing connections from which are forbidden by the firewall.

It is noteworthy that after the attacked network has been configured, all information is stored in the Network Agent’s database in the tables that are storages of examples of notions of the fragment of Network Agent’s application domain ontology (see paragraph 2.5.1).



Firewalls:

AILFirewall
AGNUS

Prohibited Attacks:

Name	Prob
SFI	1.00
SX	1.00
SN	1.00
RF	0.95
RS	0.95
IS	0.95
IDOS	0.95
PF	0.95
SA	0.95

Forbidden Local Addresses and Ranges:

210.122.25.15

210.122.25.23	210.122.25.255

Forbidden Remote Addresses and Ranges:

161.43.201.1	161.43.201.255
161.43.202.128	161.43.202.255

Fig.2.5.6. Example of a dialog window for firewall parameters configuration

2.5.3. State machines model of the Network Agent operation

The Network Agent's state machine model serves to describe the Network Agent's behavior. It specifies operations for processing incoming messages, forming the response, and sending the response message to the Hacker Agent.

The state machine model of the Network Agent is represented by a single state machine N. The presence of a single state machine is explained by the fact that the Network Agent is essentially a less complex agent compared to the Hacker Agent, and there is no need to define its next steps. The Network Agent only processes messages and forms the response.

Each of the states of the state machine model is correlated to one of the attack classes.

In the script that corresponds to that state, the following actions are executed:

- calling the firewall model;
- initialization of the network response generation methods;
- enumeration of all hosts of the network in case of a network attack (which is directed to all hosts);
- generating the outgoing messages and sending them to the Hacker Agent.

The presence of several states is conditioned by a different logic to be used by the Network Agent for processing different classes of messages that specify attack actions. The state machine N is initialized when another incoming message is received by the Network Agent.

The following is the initialization condition: `IF (NewAttack.Exist()).`

The basic examples of the notions used in the Network Agent operation are *NewAttack*, *Attack* and *xAttack*. All of them are instances of the notion *Attack*. *NewAttack* realizes the incoming message (both on the hacker side and the network side), *Attack* is the outgoing message. *xAttack* specifies a collection of examples of the outgoing message. The collection is used when a set of multiple attributes, e.g., numbers of ports, needs to be sent from one of the hosts.

An example of the initialization of the method for sending the message containing a collection (script *Net_SPIH_Do*) is as follows:

```
MESSAGE(xAttack(ALL),InformTemplate,InReplyWith=InMSG.ReplyWith);
```

If the connection between agents, which is always initiated by the Network Agent, is terminated, the latter sends a message of the following type:

```
MESSAGE (0,ReplyTemplate,InReplyWith=InMSG.ReplyWith);
```

with zero context (message without content) or a message of the type:

```
MESSAGE (Attack, ReplyTemplate, InReplyWith=InMSG.ReplyWith);
```

where only one message is sent by the Network Agent in a single session.

The state machine *N* is communicational. Auxiliary (connecting) and non-terminal states are absent in this state machine.

In each of the states, with the exception of a terminal state, the corresponding scripts are initialized, that specify operations to process incoming messages, address the program implementation of the firewall model and the network response model, and form the outgoing messages.

From the four types of scripts provided for by the MASDK state machines model (*Entry Script*, *Do Script*, *Exit Script* and *Transition Action*), the state machine *N* uses only *Do* scripts. The other scripts are simply not necessary. If there is a need in the future for increased complexity of the Network Agent operation for enhancing the prototype's possibilities, other types of scripts may also be involved. Such expansion may be necessary if, for example, several Hacker Agents take part in the attack realization, or if the Network Agent's actions need to be logged.

The operation of the state machine *N* is synchronous in relation to the Hacker Agent. At each point in time, only one incoming message from the Hacker Agent is processed. For each such message, the state machine *N* is initialized. After the response has been sent and the session has been terminated, this information is erased from RAM.

The diagram of the Network Agent's state machine *N* is shown in Fig.2.5.7.

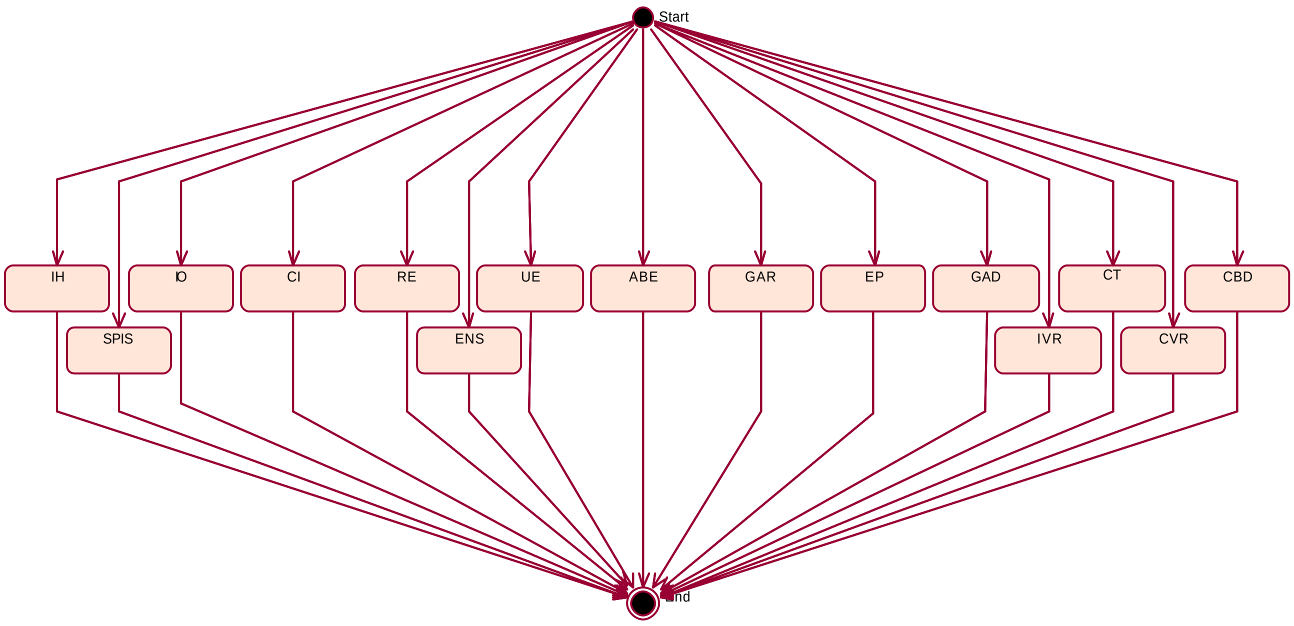


Fig.2.5.7. Diagram of the Network Agent's state machine *N*

The scripts used in each of the states are described in Appendix 2.

In the state *Start*, the firewall (network or local) is checked, after which, if the firewall is absent or does not block the hacker's attack, control is delegated to one of the states of the set of states {*IH, SPIS, IO, CI, RE, ENS, UE, ABE, GAR, EP, GAD, IVR, CVR, CT, CBD*}. The control delegation is performed depending on the attack signature and the attack belonging to one of the subclasses *SubClass0*, *SubClass1* and *SubClass2*.

The scripts use external functions *IsFirewalled* and *AttRandom*, which address the firewall model and the model of network's response to malefactor's attack action, accordingly.

The following is the template for initializing the function *IsFirewalled*:

```
IsFirewalled (newAttack.Name, newAttack.ip,
newAttack.HackerIP, newAttack.IsNet, bX, str);
```

The input parameters of the function *IsFirewalled* are as follows:

- Attack signature (*NewAttack.Name*);
- IP address of the attacked network (*newAttack.ip*) or host (*Host.IP*);
- Hacker's IP address (*newAttack.HackerIP)*;
- Logical variable that determines whether this attack is a network attack (*newAttack.IsNet*).

The output parameters of the function *IsFirewalled* are as follows:

- Logical variable *bX* that shows whether the attack is blocked by the firewall or not;
- String variable *str* (in the outgoing message *Attack.FailMessage*); if *bX=TRUE*, this variable contains the description of the reason the attack was blocked; if *bX=FALSE*, the variable contains an empty value.

The following is the template for initializing the function *AttRandom*:

```
AttRandom (newAttack.Name, newAttack.SubClass0,
newAttack.SubClass1, newAttack.SubClass2, Host.IP, bX);
```

The input parameters of the function *AttRandom* are as follows:

- attack signature (*NewAttack.Name*);
- attack subclass of 0th level (*newAttack.SubClass0*) (if one subclass is sufficient for identifying the attack by the network response generation model, other subclasses may be empty, e.g., as in the script *Net_IH_Do*);
- attack subclass of 1st level (*newAttack.SubClass1*);
- attack subclass of 2nd level (*newAttack.SubClass2*);
- IP address of the host where the response is formed – *Host.IP*.

The function *AttRandom* has only one output parameter – a logical variable *bX* that determines whether the response message will be formed.

The network response generation algorithm is described in more detail in paragraph 2.5.4.

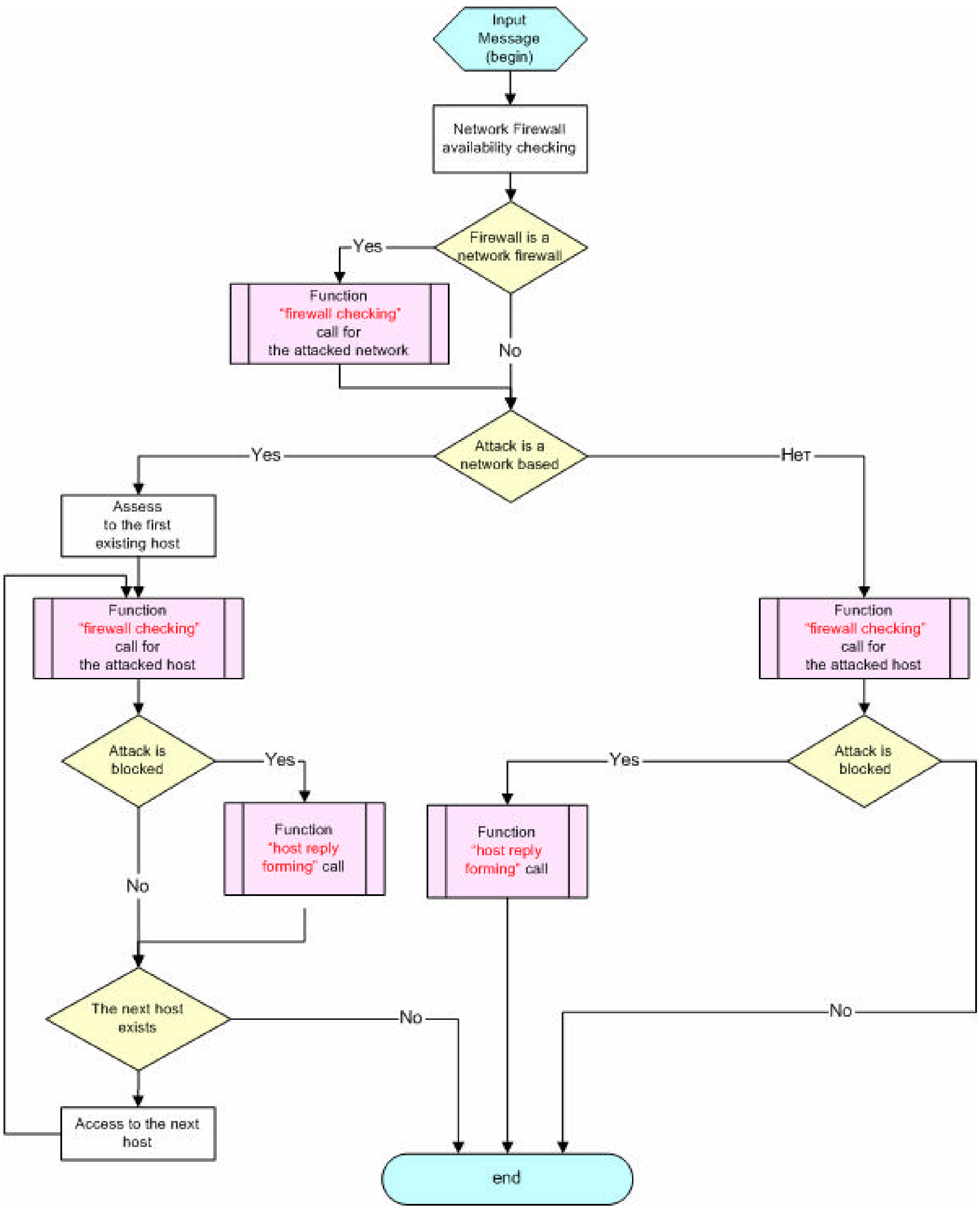
2.5.4. Component calculating the probabilities of Hacker Agent's actions success and generating network response

The program realization of the network response generation component operates with the following notion of the ontology: *Attack, LAN, Host, Firewall, ForbiddenLocalAddr, ForbiddenRemoteAddr, Attacks*.

The operation of the agent, or better to say, its state machine model, is initialized when a message that specifies the attack action is received from the Hacker Agent. The general scheme of the Network Agent's operation upon the receipt of the message is shown in Fig.2.5.8.

The network response model implemented in the Network Agent provides for several firewalls in the network (both network and local). The network response model addresses models of all assigned firewalls consecutively (in the order of their location in the flow of network traffic). The algorithm of Network Agent operation within the firewall model is shown in Fig.2.5.9.

If the condition of blocking the incoming message (attack action) is fulfilled for one of the firewalls, then the check are not performed for the remaining firewalls. A situation is possible where the network attack has successfully passed the network firewall and was later blocked by the firewall on one of the hosts.



Input Message (begin)
Network Firewall availability checking
Firewall is a network firewall
Yes
No
Function "firewall checking" call for the attacked network
Attack is a network based
Yes
Нет
Assess to the first existing host
Function "firewall checking" call for the attacked host
Function "firewall checking" call for the attacked host
Attack is blocked
Yes
No
Function "host reply forming" call
Yes
Attack is blocked
Function "host reply forming" call
The next host exists
No
No
Access to the next host
end

Fig.2.5.8. General scheme of the Network Agent's operation upon the message receipt

If the firewall name is not specified, i.e. there are no network or local firewalls, the Network Agent proceeds immediately to check for the conditions for formulating a successful network response.

Checking the network firewall of the attacked network (the value of the attribute *LAN.FirewallName*) and the personal firewall of the attacked host (the value of the attribute *Host.FirewallName*) begins with checking the sender's address (the value of the attribute *Attack.HackerIP*). Checking the address includes determining whether the address is local, as well as looping up the list of forbidden addresses and the range of addresses for that host. Depending on whether the sender's address is local or remote, the notion *ForbiddenLocalAddr* or *ForbiddenRemoteAddr* is used for these operations.

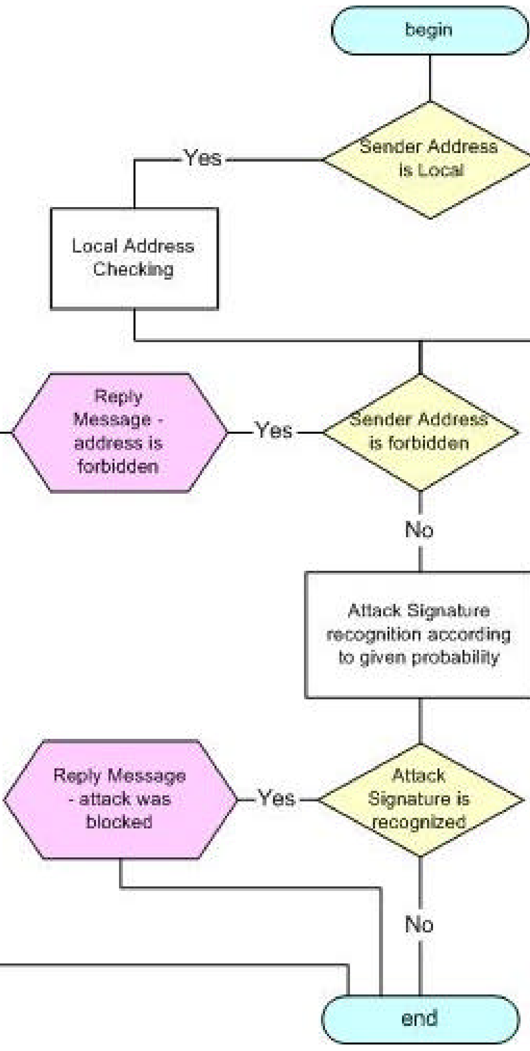

begin
Sender Address is Local
Yes
No
Local Address Checking
Remote Address Checking
Reply Message - address is forbidden
Yes
Sender Address is forbidden
No
Attack Signature recognition according to given probability
Reply Message - attack was blocked
Yes
Attack Signature is recognized
No
end

Fig.2.5.9. Algorithm of the Network Agent operation within the firewall model

If the address of the sender (Hacker Agent) is on the list or within the range of forbidden addresses for that particular firewall, the Network Agent forms a message necessarily assigning the attribute *Attack.FailMessage* the following content:

Local (Remote) Address: ***Attack.HackerIP*** *is in the forbidden IP-range;*
Blocked by Firewall LAN(Host).FirewallName.

or

Forbidden Local (Remote) Address: ***Attack.HackerIP****;*
Blocked by Firewall LAN(Host).FirewallName.

If the sender's address for that firewall is not forbidden, the Network Agent checks the signature of the attack in progress, which includes checking for that type of attacks in the firewall's database.

If that particular type of attack is found in the firewall's database, the probability of detecting the attack's signature is calculated. Based on the obtained probability value, it is determined whether the attack is detected or not.

If the attack signature is considered detected, the response message is formed, where the attribute *Attack.FailMessage* is assigned the following meaning:

Forbidden Attack ***Attack.Name****; Blocked by Firewall* ***LAN(Host).FirewallName****.*

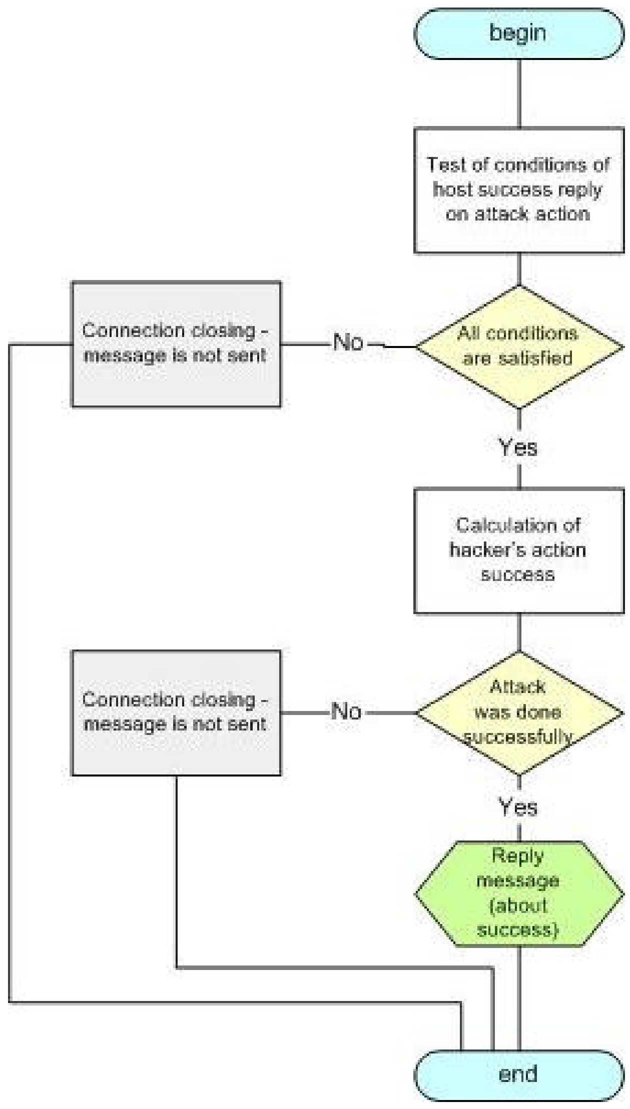

begin
Test of conditions of host success reply on attack action
All conditions are satisfied
No
Connection closing - message is not sent
Yes
Calculation of hacker's action success
Attack was done successfully
No
Connection closing - message is not sent
Yes
Reply message (about success)
end

Fig.2.5.10. Generalized algorithm for forming the host's response to the Hacker Agent's attack action

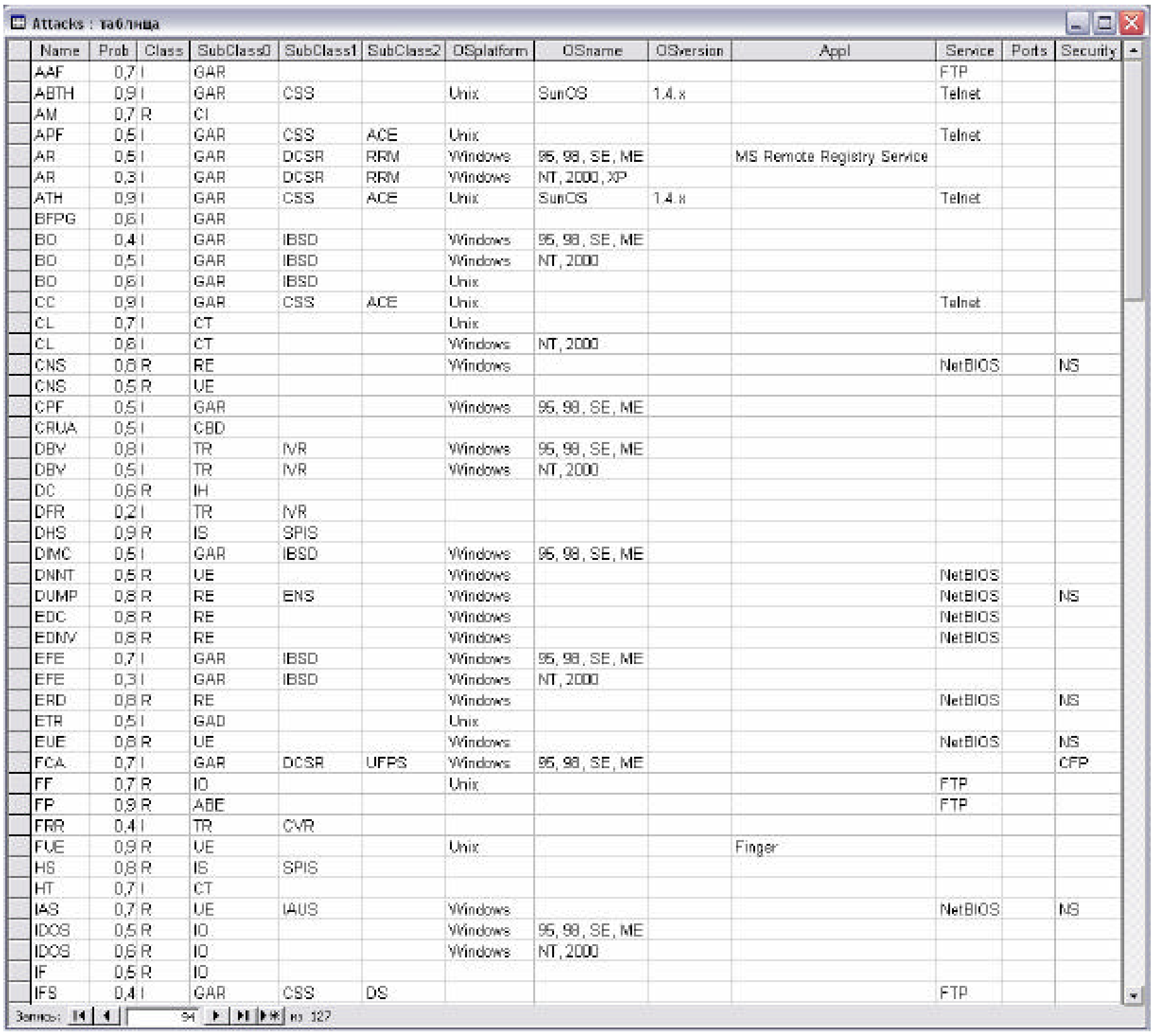
If the attack signature is not detected, then the processing for the firewall model is complete, and the Network Agent's further actions will be determined by checking all the other necessary conditions for forming the network's response. The Network Agent proceeds to these steps immediately if there are no firewalls in the network.

Further, for all the hosts of the local area network, or for a single host, if the attack is directed at a single host, the actions pertaining to forming the response are performed. Fig.2.5.10 shows a generalized algorithm for forming the host's response to the Hacker Agent's attack action.

Let us consider in more detail the procedure for checking the conditions for forming the host's response to an attack. In all further actions, the notion *Attacks* is used, which is physically mapped into the table Attacks of the Network Agent's knowledge base in regards to attacks. A fragment of the table is shown in Fig.2.5.11.

Initially, in the table Attacks, an attack *Attack.Name* with subclasses *Attack.SubClass0*, *Attack.SubClass1*, *Attack.SubClass2* (if available), specified in the incoming message, is searched. If the attack is not found, then the message exchange session is terminated by the Network Agent, and no messages are sent. The Hacker Agent classifies this as failure.

If the attack is found in table Attacks (an instance of notion *Attacks* with the name *Attack.Name* exists), then all the assigned conditions are checked. There may be a situation where none of the conditions are present.



Attacks : таблица

Name	Prob	Class	SubClass0	SubClass1	SubClass2	OSplatform	OSname	OSversion	Appl	Service	Ports	Security
AAF	0,7	I	GAR							FTP		
ABTH	0,9	I	GAR	CSS		Unix	SunOS	1.4.x		Telnet		
AM	0,7	R	CI									
APF	0,5	I	GAR	CSS	ACE	Unix				Telnet		
AR	0,5	I	GAR	DCSR	RRM	Windows	95, 98, SE, ME		MS Remote Registry Service			
AR	0,3	I	GAR	DCSR	RRM	Windows	NT, 2000, XP					
ATH	0,9	I	GAR	CSS	ACE	Unix	SunOS	1.4.x		Telnet		
BFPG	0,6	I	GAR									
BO	0,4	I	GAR	IBSD		Windows	95, 98, SE, ME					
BO	0,5	I	GAR	IBSD		Windows	NT, 2000					
BO	0,6	I	GAR	IBSD		Unix						
CC	0,9	I	GAR	CSS	ACE	Unix				Telnet		
CL	0,7	I	CT			Unix						
CL	0,6	I	CT			Windows	NT, 2000					
CNS	0,8	R	RE			Windows				NetBIOS		NS
CNS	0,5	R	UE									
CPF	0,5	I	GAR			Windows	95, 98, SE, ME					
CRUA	0,5	I	CBD									
DBV	0,8	I	TR	IVR		Windows	95, 98, SE, ME					
DBV	0,5	I	TR	IVR		Windows	NT, 2000					
DC	0,6	R	IH									
DFR	0,2	I	TR	IVR								
DHS	0,9	R	IS	SPIS								
DIMC	0,5	I	GAR	IBSD		Windows	95, 98, SE, ME					
DNNT	0,5	R	UE			Windows				NetBIOS		
DUMP	0,8	R	RE	ENS		Windows				NetBIOS		NS
EDC	0,8	R	RE			Windows				NetBIOS		
EDNV	0,8	R	RE			Windows				NetBIOS		
EFE	0,7	I	GAR	IBSD		Windows	95, 98, SE, ME					
EFE	0,3	I	GAR	IBSD		Windows	NT, 2000					
ERD	0,8	R	RE			Windows				NetBIOS		NS
ETR	0,5	I	GAD			Unix						
EUE	0,8	R	UE			Windows				NetBIOS		NS
FCA	0,7	I	GAR	DCSR	UFPS	Windows	95, 98, SE, ME					CFP
FF	0,7	R	IO			Unix				FTP		
FP	0,9	R	ABE							FTP		
FRR	0,4	I	TR	CVR								
FUE	0,9	R	UE			Unix			Finger			
HS	0,8	R	IS	SPIS								
HT	0,7	I	CT									
IAS	0,7	R	UE	IAUS		Windows				NetBIOS		NS
IDOS	0,5	R	IO			Windows	95, 98, SE, ME					
IDOS	0,6	R	IO			Windows	NT, 2000					
IF	0,5	R	IO									
IFS	0,4	I	GAR	CSS	DS					FTP		

Запись: 94 из 127

Fig.2.5.11. Fragment of the Network Agent's knowledge base about attacks

The conditions may put constraints on the operating system (*OSplatform*, *OSname*, *OSversion*), the running applications (*Appl*), installed services (*Service*), active ports (*Ports*) and the security parameters (*Security*).

If there is a condition for the host the attacker's actions address, its knowledge base is checked (notions *Host*, *Appl*, *Security*, *Service*, etc.), and if the corresponding attributes of these notions are also present in the list of conditions, then the condition is considered fulfilled.

If all the conditions are fulfilled, then the probability of the attacker's success is calculated (that value is assigned to the attribute *Prob*). After that, either the corresponding message about the success of the attack is sent, or the connection is terminated, and no response message is sent.

Depending on the information received from the Network Agent as a response message (or the absence of a response message), the Hacker Agent makes the decision in regards to its further actions to implement the threat.

2.6. Case-study Simulation: examples of Attack Simulator performance and its evaluation

The main purpose of the experiments lead within the framework of the Project has consisted in demonstration of the Attack Simulator prototype efficiency for various specifications of attacks and an attacked network configuration.

Besides, the authors of the Project had the purpose to investigate the Attack Simulator prototype opportunities for realization of the following tasks:

(1) *Checking a computer network security policy at stages of conceptual and logic design of network security system*. This task can be solved by simulation of attacks at a macro-level and researches of responding a being designed (analyzed) network model;

(2) *Checking security policy (including vulnerabilities recognition) of a real-life computer network*. This task can be solved by means of simulation of attacks at a micro-level, i.e. by generating a network traffic corresponding to real activity of malefactors on realization of various security threats.

Therefore all experiments have been divided into two classes:

(1) *Experiments on simulation of attacks on macro-level*. In these experiments, generation and investigation of malicious actions against computer network model were carried out;

(2) *Experiments on simulation of attacks on micro-level*. In these experiments, generation malicious network traffic against a real computer network was fulfilled.

The results of the lead experiments on simulation of attacks for the specified two classes of tasks are described in paragraphs 2.6.1 and 2.6.2 accordingly.

2.6.1. Simulation of attacks on macro-level (generation malicious actions against computer network model)

In the experiments on simulation of attacks on macro-level, explorations of attacks for all malefactor's intentions implemented by the Attack Simulator have been carried out (Tab.2.6.1). These experiments were carried out under various parameters of the attack task specification and an attacked computer network configuration.

Tab.2.6.1. Malefactor's intentions

Number	Designation	Interpretation
Reconnaissance (*R*)		
1	*IH*	Identification of the running Hosts
2	*IS*	Identification of the host Services
3	*IO*	Identification of the host Operating system
4	*RE*	Resource Enumeration
5	*UE*	Users and groups Enumeration
6	*ABE*	Applications and Banners Enumeration
Implantation and threat realization (*I*)		
7	*GAR*	Gaining Access to Resources
8	*EP*	Escalating Privilege
9	*CVR*	Confidentiality Violation Realization or Confidentiality destruction
10	*IVR*	Integrity Violation Realization or Integrity Destruction
11	*AVR*	Availability Violation Realization or Denial of Service
12	*CBD*	Creating Back Doors

Besides malefactor's intention, the influence of the following parameters on attacks efficacy was investigated at carrying out of experiments:

- Protection degree of Network Firewall (PNF);
- Protection degree of attacked host (Personal) Firewall (PPF);
- Protection Parameters of attacked host (PP);
- degree of hacker's Knowledge about a Network (KN).

For intention *"Reconnaissance"* we have investigated only influence of protection degree of network firewall. Three values of this parameter were used:

1 – "Strong" (if firewall can protect from 60-90% of attacks);

2 – "Medium" (if firewall can protect from 20-50% of attacks);

3 – "None" (if firewall does not protect or is absent).

For intention "*Implantation and threat realization*" we have used the following values of parameters:

- For protection degree of (network or personal) firewalls:
  1 – "Strong" (if firewall can protect from 60-90% of attacks);
  2 – "None" (if firewall does not protect or is absent).
- For protection parameters of attacked host:
  1 – "Strong" (60-90% of security parameters have secure values, for example, strong password, absence of sharing files and printers, and other resources, absence of trusted hosts, etc.);
  2 – "Weak" (security parameters are weak).
- For degree of hacker's knowledge about a network:
  1– "Good" (hacker knows about 50-80% of information about network);
  2 – "Nothing" (hacker knows nothing about network).

The assumption was accepted, that the probability of a firewall blockage for attacks which signatures are available in the firewall base is equal 0,9.

Attacks were simulated on various configurations of a computer network.

To investigate the Attack Simulator possibilities, we have selected the following *parameters of attack realization outcome*:

- *NS* (Number of attack Steps) – number of terminal level attack actions;
- *PIR* (Percentage of Intention Realization) – percentage of the hacker's intentions realized successfully (for "Reconnaissance" it is a percentage of objects about which the information has been received; for "Implantation and threat realization" it is a percentage of successful realizations of the common attack goal on all runs);
- *PAR* (Percentage of Attack Realization) – percentage of "positive" messages (responses) of the Network Agent on attack actions (the "positive" messages are designated in attack visualization window by green lines);
- *PFB* (Percentage of Firewall Blocking) – percentage of attack actions blockage by firewall (red lines in attack visualization window);
- *PRA* (Percentage of Reply Absence) – percentage of "negative" messages (responses) of the Network Agent on attack actions (gray lines in attack visualization window).

For each separate experiment, various realizations of attacks (runs) were carried out. In experiments described in this paragraph, three realizations (runs) with identical initial data were carried out. The results received on each experiment, were averaged.

Taking into account limitation of the Report volume, we shall describe results of experiments only for two classes of intentions concerning to each of the high-level intentions *Reconnaissance* (*R*) and *Implantation and threat realization* (*I*).

For high-level intention *R* we shall present the results of experiments for intentions *Identification of the host Services* (IS) and *Applications and Banners Enumeration* (ABE), and for high-level intention *I* – the results of experiments for intentions *Gaining Access to Resources* (GAR) and *Confidentiality Violation Realization or Confidentiality destruction* (CVR).

In described experiments it was supposed, that the researched computer network has "star" structure and includes five hosts. Main parameters of hosts are submitted in Tab.2.6.2. Names of table columns correspond to the attributes of the ontology notion ***Host***.

Tab.2.6.2. Malefactor's intentions

DomName	IP	OSversion	Mask	SysTime	IsDns	OStype	OSplatform
spiiran-erv.ail.net	192.168.130.135	SP3	255.255.255.224	12:25:16.7754	T	2000	Windows
Vladimir.lan3.net	192.168.130.138	SE	255.255.255.224	12:25:04.5891	F	98	Windows
Oleg.lan3.net	192.168.130.139	SP1	255.255.255.224	12:25:08.0126	F	2000	Windows
Victor.lan3.net	192.168.130.140	7	255.255.255.224	12:25:12.9800	F	Linux Mandrace	Unix
Igor.lan3.net	192.168.130.141	1.4.x	255.255.255.224	12:25:22.4511	F	SunOS	Unix

1. Description of experiments for high-level intention *Reconnaissance* (*R*)

For all intentions of high-level intention R, attacks have been directed on all hosts of a network. It was supposed, that the hacker knows only the IP-address of the attacked network.

Attacks were carried out for three gradation of protection degree of network firewall: 1 - "Strong"; 2 - "Medium"; 3 - "None".

Results of the fulfilled experiments for intentions *Identification of the host Services* (IS) and *Applications and Banners Enumeration* (ABE) are depicted in Tab.2.6.3.

1.1. Description of experiments for intention Identification of the host Services (IS)

At realization of intention IS the following terminal level attacks are generated:

ST, SS, SFI, SX, SN, SU, HS, SFB, DHS, PS,

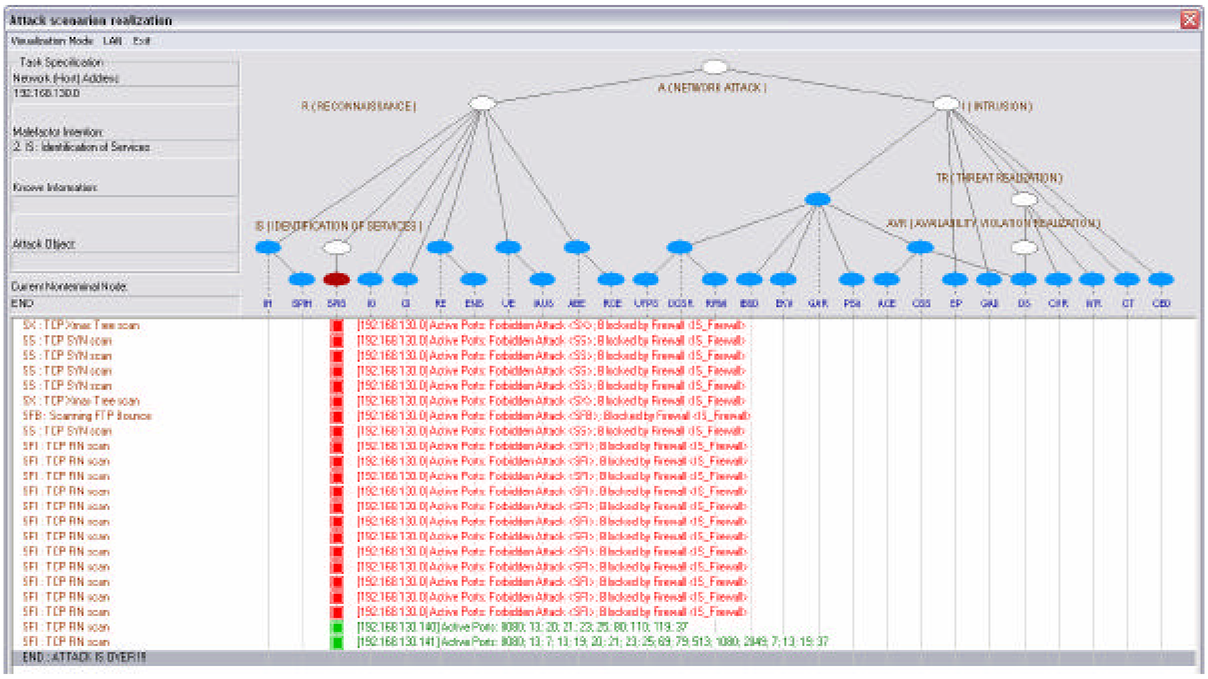
where ST – “`TCP connect scan`”, SS – “`TCP SYN scan`”, SFI – “`TCP FIN scan`”, SX – “`TCP Xmas Tree scan`”, SN – “`TCP Null scan`”, SU – “`UDP scan`”, HS – “`Half scan`”, SFB – “`Scanning “FTP Bounce”`”, DHS – “`Dumb host scan`”, PS – “`“Proxy”-scanning`”.

At carrying out the attacks realizing intention IS, it was supposed, that depending on its protection degree a network firewall can block the following terminal level attacks:

1) For “Strong”: ST, SS, SFI, SX, SN, SU, SFB, PS;
2) For “Medium”: ST, SS, SFI, SX;
3) For “None”: - .

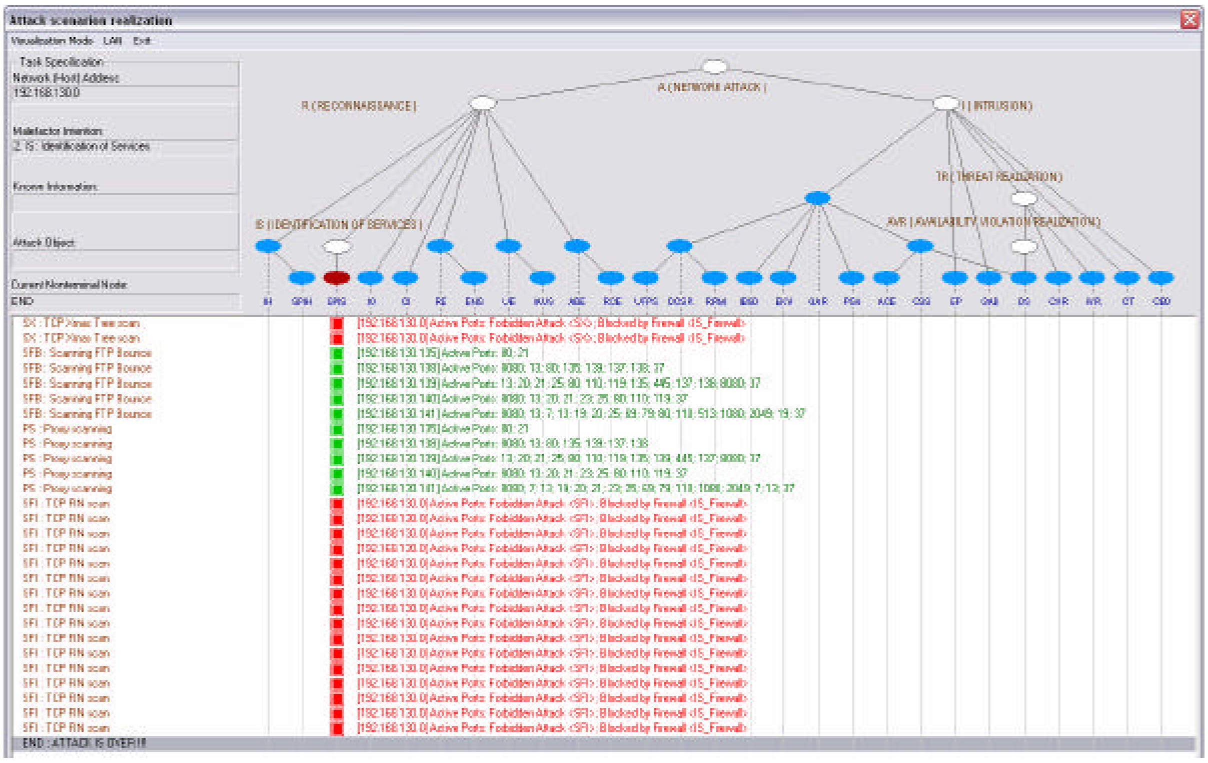
Examples of the screens displaying the attack generation processes under the intention IS realization for various protection degrees of network firewall are submitted in Fig.2.6.1 and Fig.2.6.2.

Examples of the screens displaying the fragments of attack traces logs for intention IS under “Strong” (1) and “Medium” (2) protection degree of network firewall, are depicted in Fig.2.6.3 and Fig.2.6.4.


Attack scenarios realization
Visualization Mode   LAN   Exit
Task Specification
Network (Host) Address
192.168.130.0
Malefactor Intention:
2. IS: Identification of Services
Known Information:
Attack Object:
Current Nonterminal Node:
END
A (NETWORK ATTACK)
R (RECONNAISSANCE)
I (INTRUSION)
TR (THREAT REALIZATION)
IS (IDENTIFICATION OF SERVICES)
AVR (AVAILABILITY VIOLATION REALIZATION)
IH   SPIH   SRS   ID   OI   RE   ENS   UE   IAUS   ABE   ROE   UFPS   DOSR   RPBM   BSD   EKV   GAR   PSI   ACE   CSS   EP   GAI   DS   CRR   WR   CT   CBD

```
SX : TCP Xmas Tree scan    [192.168.130.0] Active Ports: Forbidden Attack <SX>; Blocked by Firewall (IS_Firewall)
SS : TCP SYN scan          [192.168.130.0] Active Ports: Forbidden Attack <SS>; Blocked by Firewall (IS_Firewall)
SS : TCP SYN scan          [192.168.130.0] Active Ports: Forbidden Attack <SS>; Blocked by Firewall (IS_Firewall)
SS : TCP SYN scan          [192.168.130.0] Active Ports: Forbidden Attack <SS>; Blocked by Firewall (IS_Firewall)
SS : TCP SYN scan          [192.168.130.0] Active Ports: Forbidden Attack <SS>; Blocked by Firewall (IS_Firewall)
SX : TCP Xmas Tree scan    [192.168.130.0] Active Ports: Forbidden Attack <SX>; Blocked by Firewall (IS_Firewall)
SFB: Scanning FTP Bounce   [192.168.130.0] Active Ports: Forbidden Attack <SFB>; Blocked by Firewall (IS_Firewall)
SS : TCP SYN scan          [192.168.130.0] Active Ports: Forbidden Attack <SS>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.0] Active Ports: Forbidden Attack <SFI>; Blocked by Firewall (IS_Firewall)
SFI : TCP FIN scan         [192.168.130.140] Active Ports: 8080; 13; 20; 21; 23; 25; 80; 110; 119; 37
SFI : TCP FIN scan         [192.168.130.141] Active Ports: 8080; 13; 7; 13; 19; 20; 21; 23; 25; 69; 79; 513; 1080; 2049; 7; 13; 19; 37
END : ATTACK IS OVER!!!
```

Fig.2.6.1. Example of the screen displaying the attack scenario generation processes of the intention IS under “Strong” (1) protection degree of network firewall


Attack scenarios realization
Visualization Mode
LAN
Exit
A (NETWORK ATTACK)
R (RECONNAISSANCE)
I (INTRUSION)
TR (THREAT REALIZATION)
IS (IDENTIFICATION OF SERVICES)
AVR (AVAILABILITY VIOLATION REALIZATION)

```
Task Specification
Network (Host) Address:
192.168.130.0
Malefactor Intention:
2. IS : Identification of Services
Known Information:
Attack Object:
Current Nonterminal Node:
END
```

```
SX : TCP Xmas Tree scan      [192.168.130.0] Active Ports: Forbidden Attack <SX> ; Blocked by Firewall <IS_Firewall>
SX : TCP Xmas Tree scan      [192.168.130.0] Active Ports: Forbidden Attack <SX> ; Blocked by Firewall <IS_Firewall>
SFB : Scanning FTP Bounce    [192.168.130.135] Active Ports: 80; 21
SFB : Scanning FTP Bounce    [192.168.130.138] Active Ports: 8080; 13; 80; 135; 139; 137; 138; 37
SFB : Scanning FTP Bounce    [192.168.130.139] Active Ports: 13; 20; 21; 25; 80; 110; 119; 135; 445; 137; 138; 8080; 37
SFB : Scanning FTP Bounce    [192.168.130.140] Active Ports: 8080; 13; 20; 21; 23; 25; 80; 110; 119; 37
SFB : Scanning FTP Bounce    [192.168.130.141] Active Ports: 8080; 13; 7; 13; 19; 20; 25; 69; 79; 80; 111; 513; 1080; 2049; 19; 37
PS : Proxy scanning          [192.168.130.135] Active Ports: 80; 21
PS : Proxy scanning          [192.168.130.138] Active Ports: 8080; 13; 80; 135; 139; 137; 138
PS : Proxy scanning          [192.168.130.139] Active Ports: 13; 20; 21; 25; 80; 110; 119; 135; 139; 445; 137; 8080; 37
PS : Proxy scanning          [192.168.130.140] Active Ports: 8080; 13; 20; 21; 23; 25; 80; 110; 119; 37
PS : Proxy scanning          [192.168.130.141] Active Ports: 8080; 7; 13; 19; 20; 21; 23; 25; 69; 79; 111; 1080; 2049; 7; 13; 37
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
SFI : TCP FIN scan           [192.168.130.0] Active Ports: Forbidden Attack <SFI> ; Blocked by Firewall <IS_Firewall>
END : ATTACK IS OVER!!!
```

Fig.2.6.2. Example of the screen displaying the attack scenario generation processes of the intention IS under “Medium” (2) protection degree of network firewall

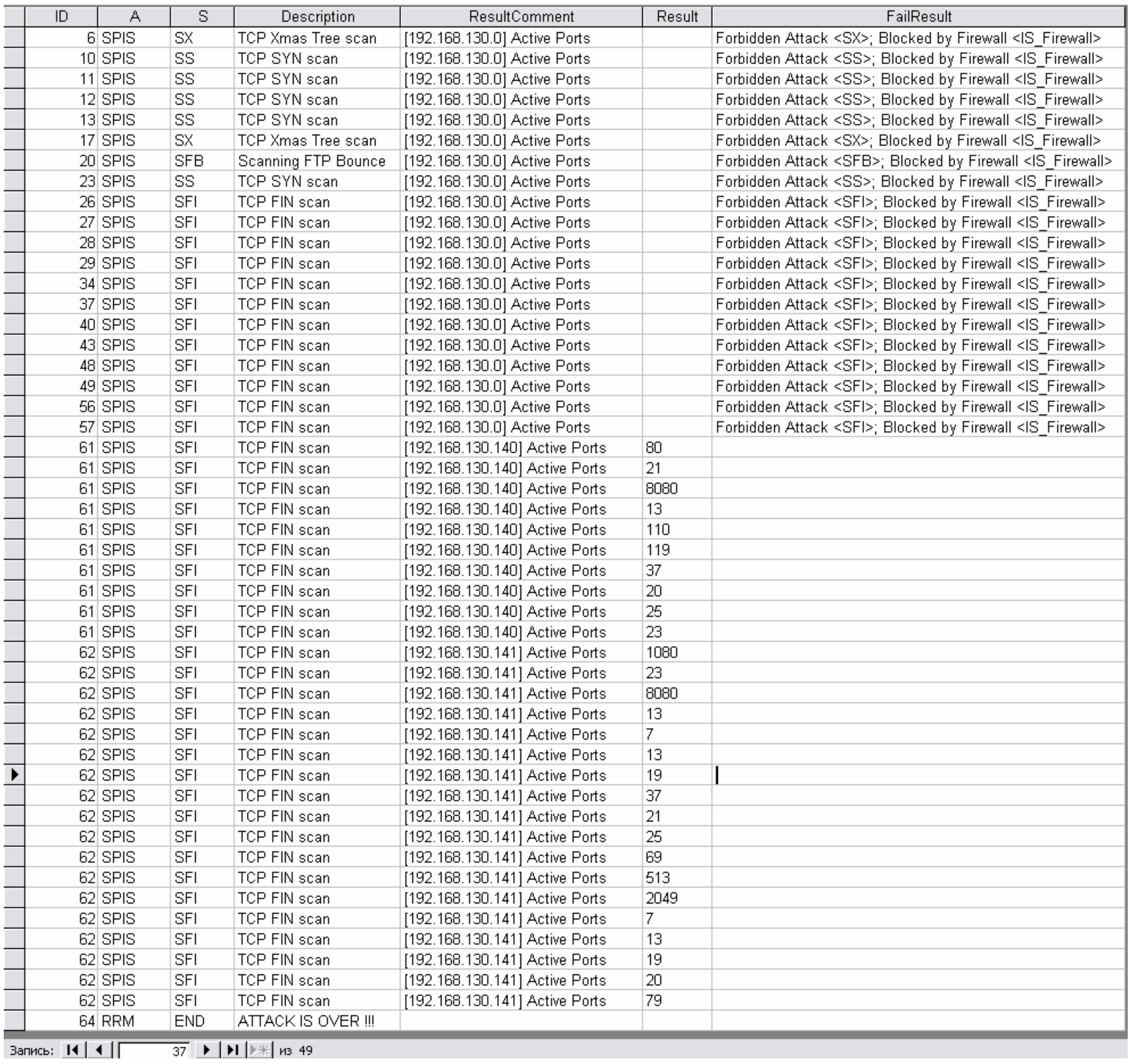
The attributes of the logs are as follows (they correspond to the attributes of the ontology notions ***Log*** and ***LogResult***):

- ***ID*** – a unique number identifying the state of a state machine;
- ***A*** – state machine name;
- ***S*** – the used state of a state machine;
- ***Description*** – description of the state machine’s state (except for the intermediate states); if the state is terminal, then the action description is specified; if it is non-terminal, then the description of attack class is recorded;
- ***ResultComment*** – the description of the result that can be obtained in the used state *S* (if that state is terminal);
- ***Result*** – information received from the host or message about the successful attack in the terminal state;
- ***FailResult*** – information received from the attacked network in case the attacked is blocked by a firewall.

Graphical representation of attack realization outcome parameters (NS, PIR, PAR, PFB, PRA) values at realization of intention IS depending on values of protection degree of network firewall is displayed in Fig.2.6.5.

The integral diagram showing the values of main outcome parameters at intention IS realization is depicted in Fig.2.6.6.

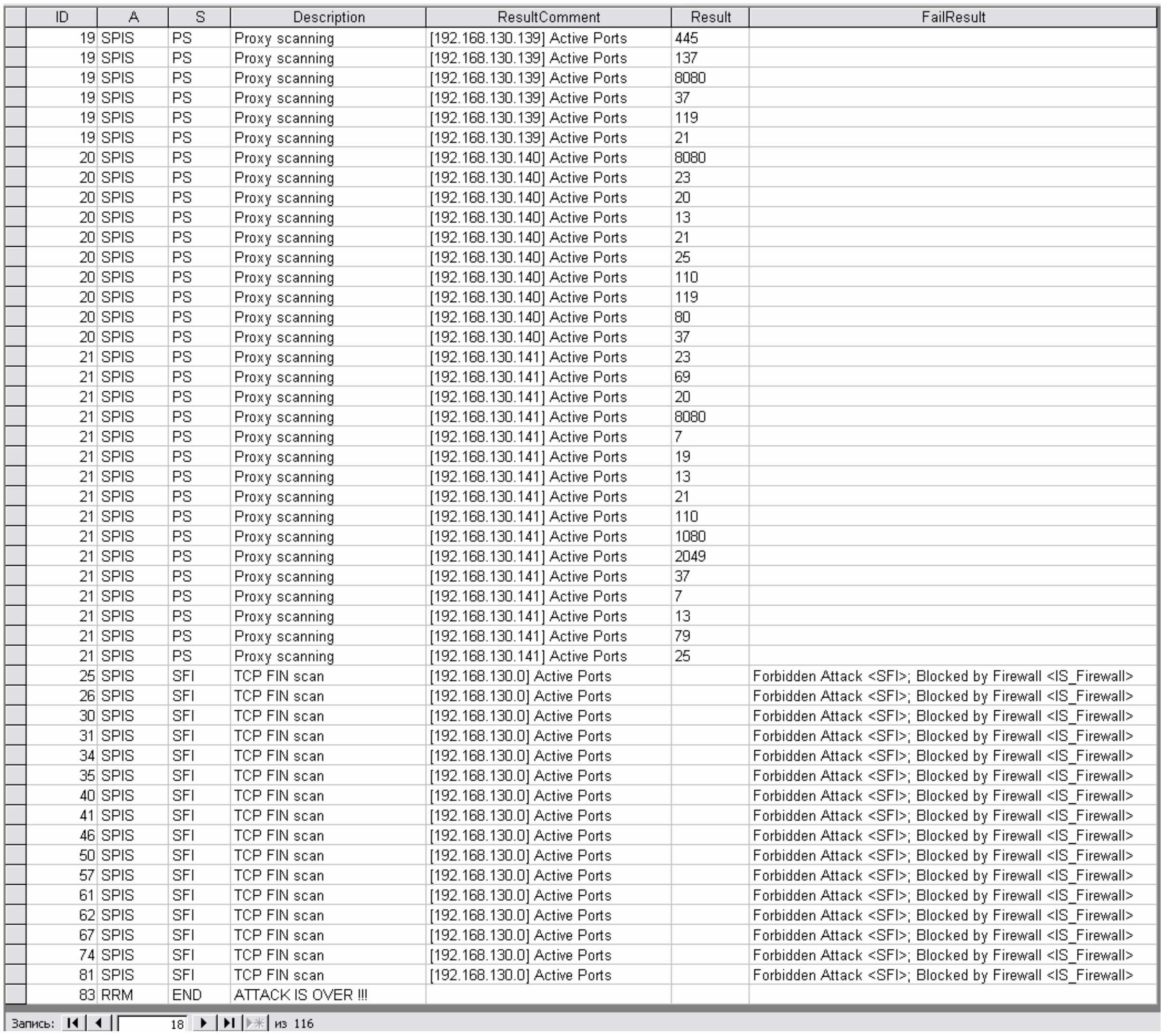
Changes of parameters PIR, PAR, PFB, and PRA for various network firewall configurations are represented in Fig.2.6.7 as graphic dependences.



ID	A	S	Description	ResultComment	Result	FailResult
6	SPIS	SX	TCP Xmas Tree scan	[192.168.130.0] Active Ports		Forbidden Attack <SX>; Blocked by Firewall <IS_Firewall>
10	SPIS	SS	TCP SYN scan	[192.168.130.0] Active Ports		Forbidden Attack <SS>; Blocked by Firewall <IS_Firewall>
11	SPIS	SS	TCP SYN scan	[192.168.130.0] Active Ports		Forbidden Attack <SS>; Blocked by Firewall <IS_Firewall>
12	SPIS	SS	TCP SYN scan	[192.168.130.0] Active Ports		Forbidden Attack <SS>; Blocked by Firewall <IS_Firewall>
13	SPIS	SS	TCP SYN scan	[192.168.130.0] Active Ports		Forbidden Attack <SS>; Blocked by Firewall <IS_Firewall>
17	SPIS	SX	TCP Xmas Tree scan	[192.168.130.0] Active Ports		Forbidden Attack <SX>; Blocked by Firewall <IS_Firewall>
20	SPIS	SFB	Scanning FTP Bounce	[192.168.130.0] Active Ports		Forbidden Attack <SFB>; Blocked by Firewall <IS_Firewall>
23	SPIS	SS	TCP SYN scan	[192.168.130.0] Active Ports		Forbidden Attack <SS>; Blocked by Firewall <IS_Firewall>
26	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
27	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
28	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
29	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
34	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
37	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
40	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
43	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
48	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
49	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
56	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
57	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	80	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	21	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	8080	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	13	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	110	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	119	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	37	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	20	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	25	
61	SPIS	SFI	TCP FIN scan	[192.168.130.140] Active Ports	23	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	1080	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	23	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	8080	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	13	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	7	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	13	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	19	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	37	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	21	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	25	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	69	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	513	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	2049	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	7	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	13	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	19	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	20	
62	SPIS	SFI	TCP FIN scan	[192.168.130.141] Active Ports	79	
64	RRM	END	ATTACK IS OVER !!!			

Запись: 37 из 49

Fig.2.6.3. Example of the screen showing the fragments of attack traces logs for intention IS under “Strong” (1) protection degree of network firewall



ID	A	S	Description	ResultComment	Result	FailResult
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	445	
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	137	
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	8080	
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	37	
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	119	
19	SPIS	PS	Proxy scanning	[192.168.130.139] Active Ports	21	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	8080	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	23	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	20	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	13	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	21	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	25	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	110	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	119	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	80	
20	SPIS	PS	Proxy scanning	[192.168.130.140] Active Ports	37	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	23	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	69	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	20	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	8080	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	7	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	19	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	13	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	21	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	110	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	1080	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	2049	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	37	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	7	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	13	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	79	
21	SPIS	PS	Proxy scanning	[192.168.130.141] Active Ports	25	
25	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
26	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
30	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
31	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
34	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
35	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
40	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
41	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
46	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
50	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
57	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
61	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
62	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
67	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
74	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
81	SPIS	SFI	TCP FIN scan	[192.168.130.0] Active Ports		Forbidden Attack <SFI>; Blocked by Firewall <IS_Firewall>
83	RRM	END	ATTACK IS OVER !!!			

Запись: 18 из 116

Fig.2.6.4. Example of the screen showing the fragments of attack traces logs for intention IS under “Medium” (2) protection degree of network firewall

Tab.2.6.3. Results of experiments for intentions IS and ABE

Intention	Protection degree of Network Firewall (PNF)	First run results					Second run results					Third run results					Average results				
		NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA
IS	1	22	**10**	9	91	0	28	**20**	31	69	0	27	**60**	70	30	0	22	**30**	37	63	0
	2	31	**60**	81	18	1	28	**40**	36	64	0	49	**100**	91	7	2	31	**67**	69	30	1
	3	115	**100**	97	0	3	122	**100**	99	0	1	101	**100**	98	0	2	115	**100**	98	0	2
ABE	1	52	**80**	46	23	31	47	**60**	38	36	26	43	**100***	37	30	33	47	**80**	40	30	30
	2	41	**100**	46	25	29	26	**100**	39	23	38	65	**60**	29	15	46	44	**80**	38	22	40
	3	110	**100**	55	0	45	72	**100**	51	0	49	121	**100**	57	0	43	101	**100**	54	0	46

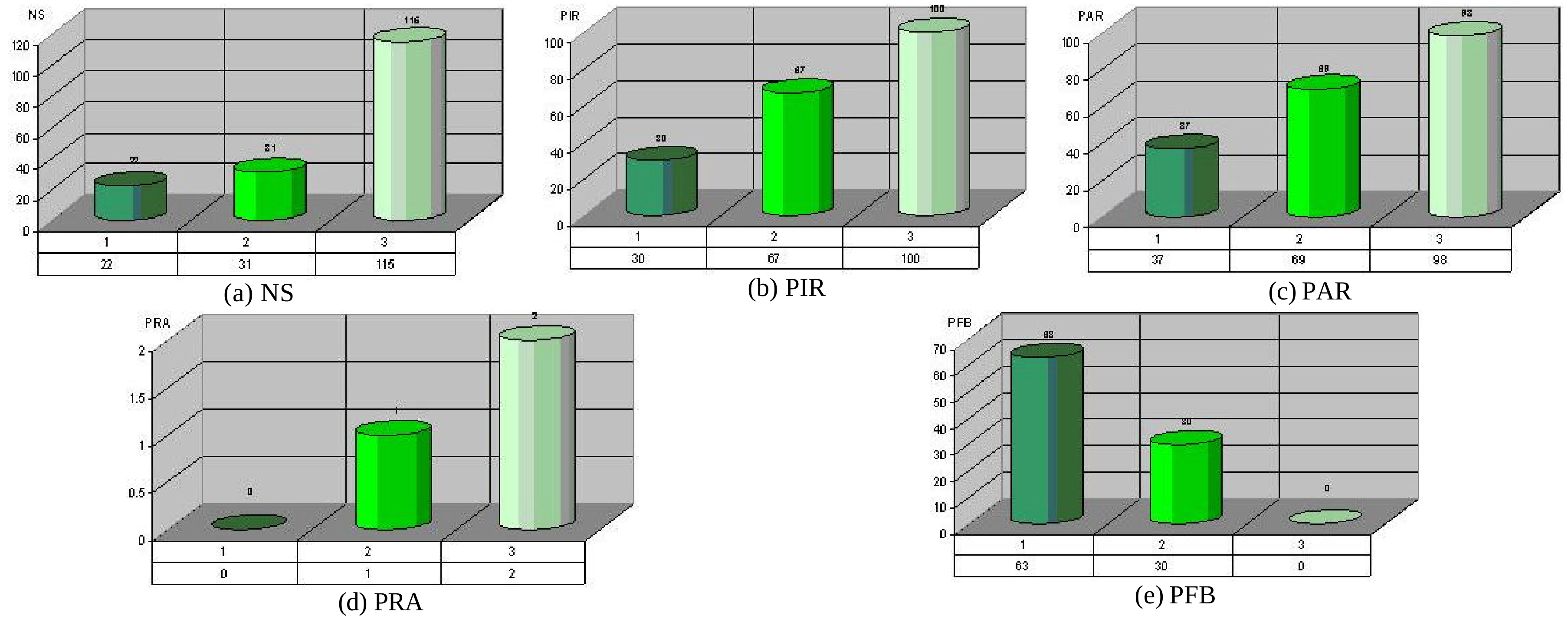


Fig.2.6.5. Graphical representation of experiments results for intention IS

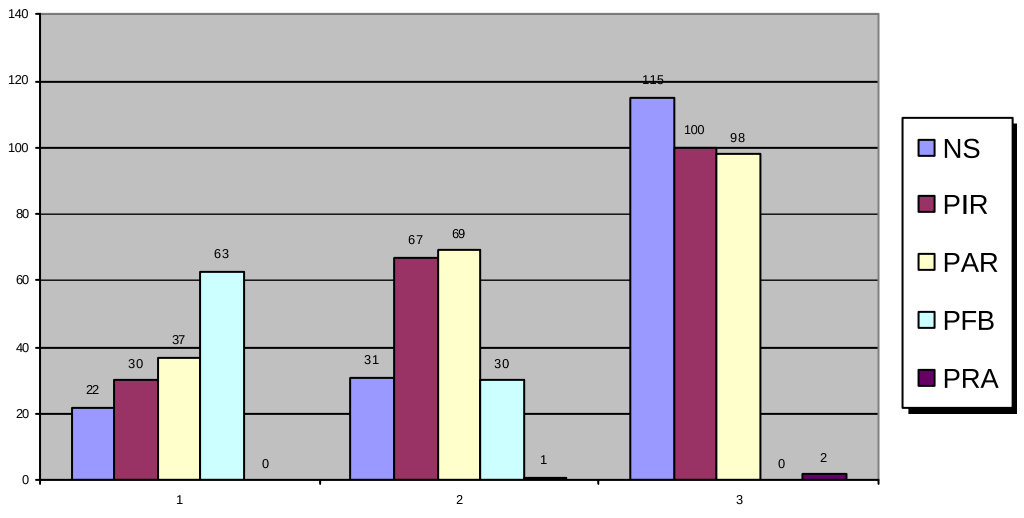


Fig.2.6.6. Integral diagram of attack outcome parameters values for intention IS

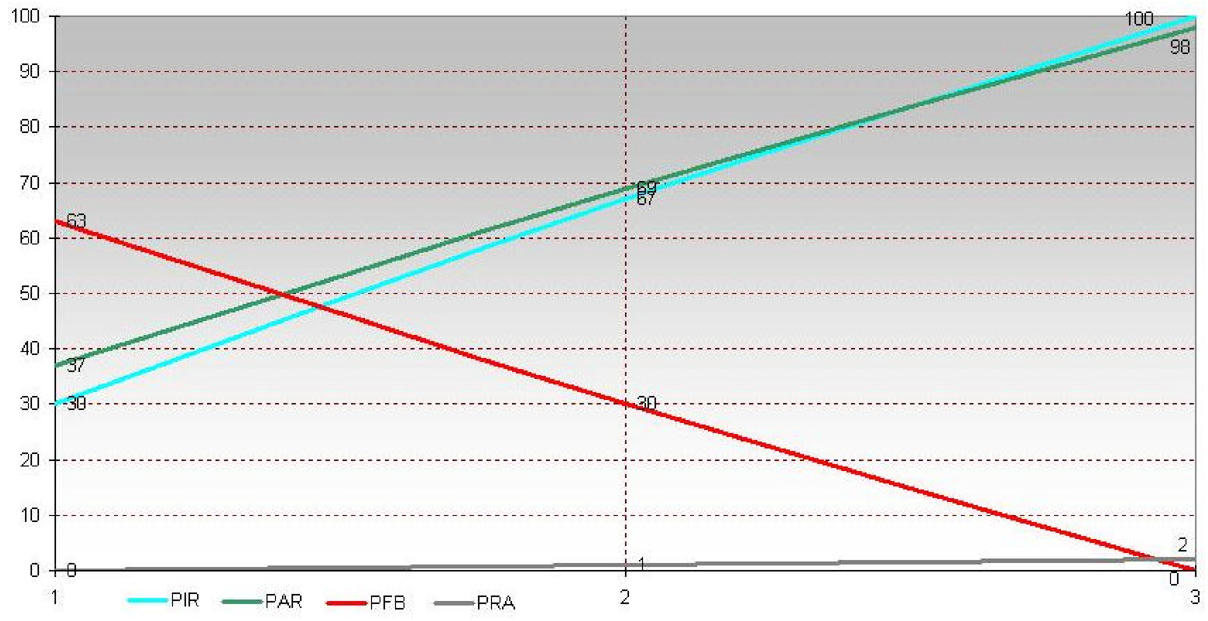


Fig.2.6.7. Changes of parameters PIR, PAR, PFB, PRA values for various network firewall configurations under realization of intention IS

1.2. Description of experiments for intention Applications and Banners Enumeration (ABE)

At realization of intention ABE the following terminal level attacks are generated:

TCBG, UNU, FP, UREG, UDUM,

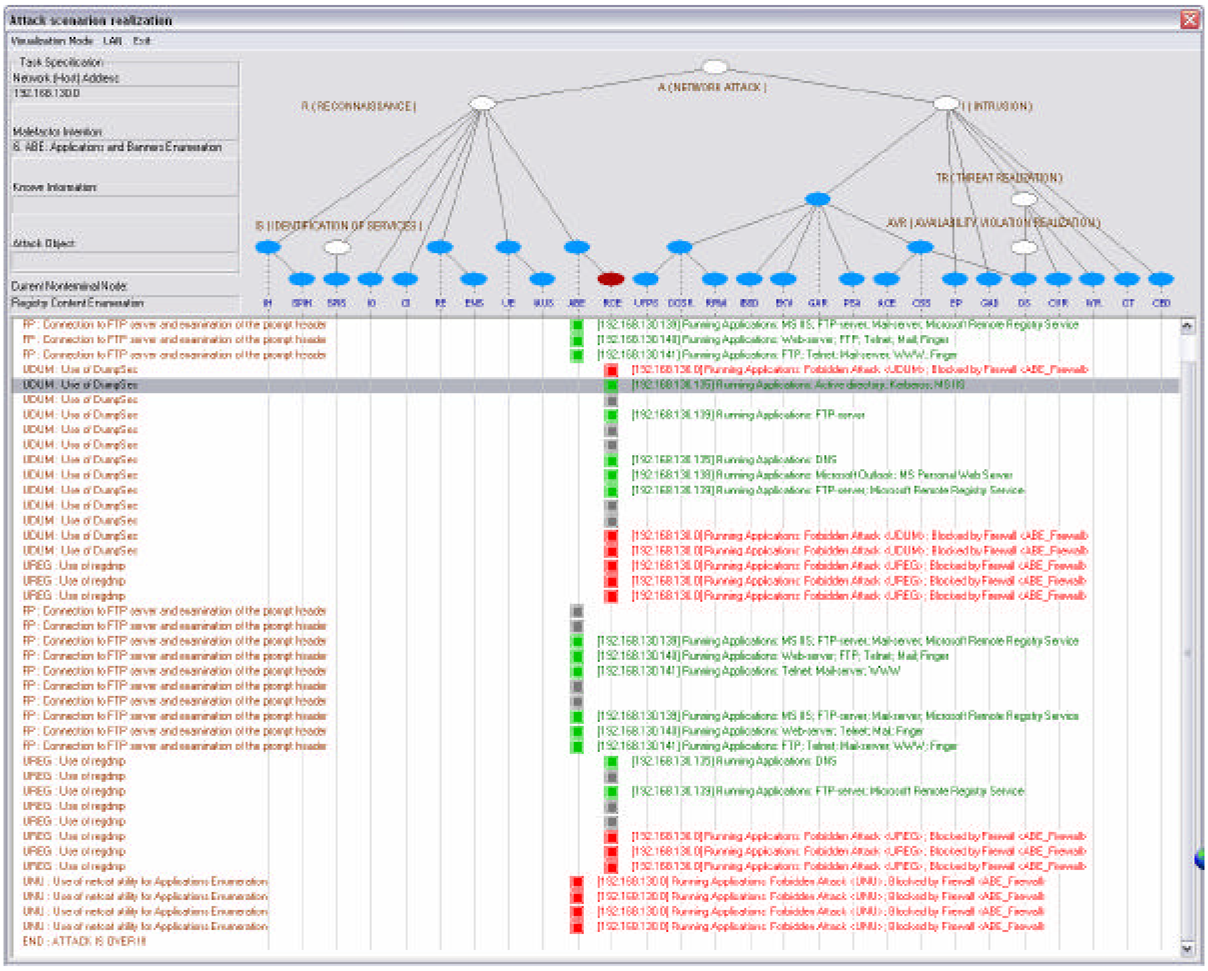
where TCBG – “`Telnet Connection Banner Grabbing`”, UNU – “`Use of` ***`netcat`*** `utility for application enumeration`”, FP – “`FTP-server prompt`”, UREG – “`Use of` ***`regdmp`***”, UDUM – “`Use of DumpSec (DumpACL) for output of all services and drivers executed on a host`”.

At carrying out the attacks realizing intention ABE, it was supposed, that depending on its protection degree a network firewall can block the following terminal level attacks:

1) For “Strong”: TCBG, UNU, UREG, UDUM;

2) For “Medium”: TCBG, UNU;

3) For “None”: - .


Attack scenario realization
Visualization Mode LAN Exit
A (NETWORK ATTACK)
R (RECONNAISSANCE)
I (INTRUSION)
TR (THREAT REALIZATION)
AVR (AVAILABILITY VIOLATION REALIZATION)
IS (IDENTIFICATION OF SERVICES)
IH
SPIN
SNS
IO
CI
RE
EMS
UE
NUS
ABE
RCE
UFPS
DOSR
RPM
BSD
BCV
GAR
PSA
ACE
CSS
EP
GAI
DS
CBR
WR
CT
CBD

Task Specification
Network (Host) Address
192.168.130.0
Malefactor Intention
6. ABE : Applications and Banners Enumeration
Known Information
Attack Object
Current Nonterminal Node
Registry Content Enumeration

```
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.139] Running Applications: MS IIS; FTP-server; Mail-server; Microsoft Remote Registry Service
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.140] Running Applications: Web-server; FTP; Telnet; Mail; Finger
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.141] Running Applications: FTP; Telnet; Mail-server; WWW; Finger
UDUM : Use of DumpSec      🟥 [192.168.138.0] Running Applications: Forbidden Attack <UDUM> ; Blocked by Firewall <ABE_Firewall>
UDUM : Use of DumpSec      🟩 [192.168.138.135] Running Applications: Active directory; Kerberos; MS IIS
UDUM : Use of DumpSec      ■
UDUM : Use of DumpSec      🟩 [192.168.138.139] Running Applications: FTP-server
UDUM : Use of DumpSec      ■
UDUM : Use of DumpSec      ■
UDUM : Use of DumpSec      🟩 [192.168.138.135] Running Applications: DNS
UDUM : Use of DumpSec      🟩 [192.168.138.138] Running Applications: Microsoft Outlook; MS Personal Web Server
UDUM : Use of DumpSec      🟩 [192.168.138.139] Running Applications: FTP-server; Microsoft Remote Registry Service
UDUM : Use of DumpSec      ■
UDUM : Use of DumpSec      ■
UDUM : Use of DumpSec      🟥 [192.168.138.0] Running Applications: Forbidden Attack <UDUM> ; Blocked by Firewall <ABE_Firewall>
UDUM : Use of DumpSec      🟥 [192.168.138.0] Running Applications: Forbidden Attack <UDUM> ; Blocked by Firewall <ABE_Firewall>
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
FP : Connection to FTP server and examination of the prompt header   ■
FP : Connection to FTP server and examination of the prompt header   ■
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.139] Running Applications: MS IIS; FTP-server; Mail-server; Microsoft Remote Registry Service
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.140] Running Applications: Web-server; FTP; Telnet; Mail; Finger
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.141] Running Applications: Telnet; Mail-server; WWW
FP : Connection to FTP server and examination of the prompt header   ■
FP : Connection to FTP server and examination of the prompt header   ■
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.139] Running Applications: MS IIS; FTP-server; Mail-server; Microsoft Remote Registry Service
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.140] Running Applications: Web-server; Telnet; Mail; Finger
FP : Connection to FTP server and examination of the prompt header   🟩 [192.168.130.141] Running Applications: FTP; Telnet; Mail-server; WWW; Finger
UREG : Use of regdmp       🟩 [192.168.138.135] Running Applications: DNS
UREG : Use of regdmp       ■
UREG : Use of regdmp       🟩 [192.168.138.139] Running Applications: FTP-server; Microsoft Remote Registry Service
UREG : Use of regdmp       ■
UREG : Use of regdmp       ■
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
UREG : Use of regdmp       🟥 [192.168.138.0] Running Applications: Forbidden Attack <UREG> ; Blocked by Firewall <ABE_Firewall>
UNU : Use of netcat utility for Applications Enumeration   🟥 [192.168.130.0] Running Applications: Forbidden Attack <UNU> ; Blocked by Firewall <ABE_Firewall>
UNU : Use of netcat utility for Applications Enumeration   🟥 [192.168.130.0] Running Applications: Forbidden Attack <UNU> ; Blocked by Firewall <ABE_Firewall>
UNU : Use of netcat utility for Applications Enumeration   🟥 [192.168.130.0] Running Applications: Forbidden Attack <UNU> ; Blocked by Firewall <ABE_Firewall>
UNU : Use of netcat utility for Applications Enumeration   🟥 [192.168.130.0] Running Applications: Forbidden Attack <UNU> ; Blocked by Firewall <ABE_Firewall>
END : ATTACK IS OVER!!!
```

Fig.2.6.8. Example of the screen displaying the attack scenario generation processes of the intention ABE under "Strong" (1) protection degree of network firewall

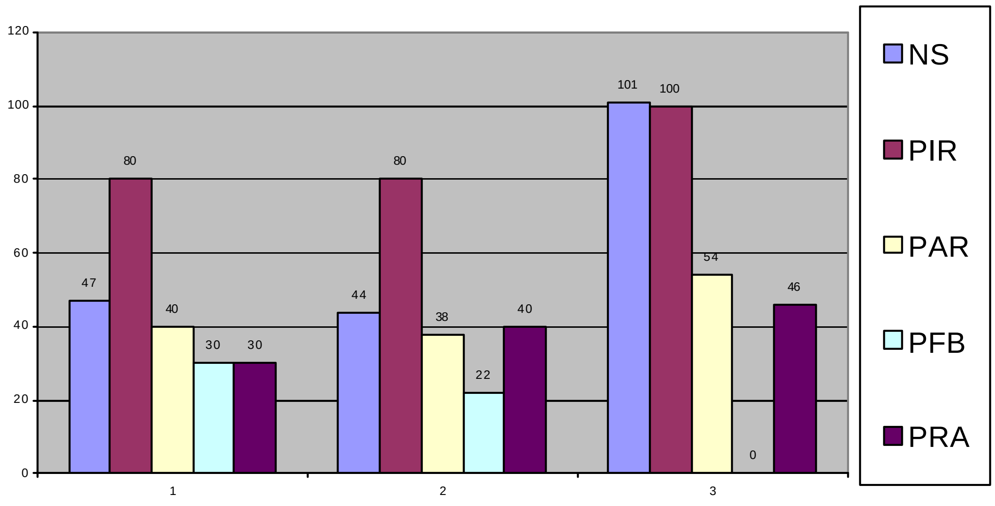


Fig.2.6.9. Integral diagram of attack outcome parameters values for intention ABE

Example of the screen displaying the attack generation processes of the intention ABE realization for "Strong" protection degree of network firewall is submitted in Fig.2.6.8.

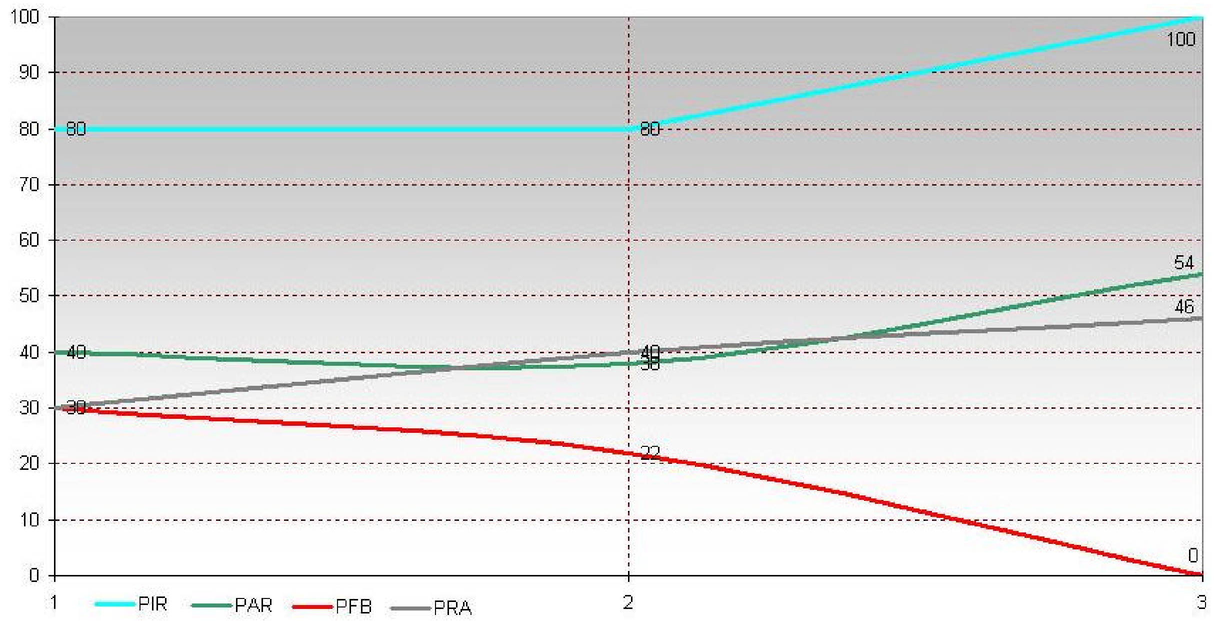


Fig.2.6.10. Changes of parameters PIR, PAR, PFB, PRA values for various network firewall configurations under realization of intention ABE

Graphical representation of attack realization outcome parameters (NS, PIR, PAR, PFB, PRA) values at realization of intention ABE depending on values of protection degree of network firewall is displayed in Fig.2.6.9.

Changes of parameters PIR, PAR, PFB, and PRA for various network firewall configurations are represented in Fig.2.6.10 as graphic dependences.

2. Description of experiments for high-level intention *"Implantation and threat realization"* (*I*)

For all intentions of high-level intention I, attacks have been directed on single host of a network.

Attacks were carried out under the following varying conditions:

(1) for two protection degrees of network firewall (1 – "Strong"; 2 – "None");

(2) for two protection degrees of personal firewall (1 – "Strong"; 2 – "None");

(3) for two degrees of protection parameters of attacked host (1 – "Strong"; 2 – " Weak");

(4) for two degrees of hacker's knowledge about a network (1 – "Good"; 2 – "Nothing").

Results of the fulfilled experiments for intentions *Gaining Access to Resources* (*GAR*) and *Confidentiality Violation Realization* (CVR) are depicted in Tab.2.6.4.

2.1. Description of experiments for intention Gaining Access to Resources (*GAR*)

Let us consider the input parameters which influence on efficacy of attacks was investigated at carrying out experiments on intention GAR realization.

1. Firewall parameters.

At realization of intention GAR, besides intention GAR, some other intentions are used. These additional intentions (IH, IS, IO, CI, RE, UE, ABE) are for getting information about an attacked network to fulfill the attacks of class GAR.

Let us consider the terminal attacks which are generated at realization of all these intentions.

Terminal attacks of intention IH (Identification of the running Hosts):

STIH, SSIH, DC,

where STIH – "`TCP connect scan`", SSIH – "`TCP SYN scan`", DC – "`Network Ping Sweeps`".

Tab.2.6.4. Results of experiments for intentions GAR and CVR

Intention	Protection degree of Network Firewall (PNF)	Protection degree of Personal Firewall (PPF)	Protection Parameters of attacked host (PP)	Degree of hacker's Knowledge about a Network (KN)	First run results					Second run results					Third run results					Average results				
					NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA	NS	PIR	PAR	PFB	PRA
GAR	1	1	1	1	139	**0**	5	36	59	112	**0**	5	35	60	120	**0**	4	40	56	124	**0**	5	37	58
				2	152	**0**	3	35	62	84	**0**	4	33	63	136	**0**	5	38	57	124	**0**	4	35	61
			2	1	141	**0**	7	36	57	148	**0**	5	49	46	130	**0**	3	49	48	140	**0**	5	45	50
				2	149	**0**	2	38	60	168	**0**	1	48	51	155	**0**	3	51	46	157	**0**	2	46	52
		2	1	1	101	**0**	21	26	53	120	**100**	17	25	58	125	**0**	4	30	66	85	**33**	14	27	59
				2	136	**0**	2	27	71	167	**0**	2	26	72	167	**100**	5	28	67	157	**33**	3	27	70
			2	1	135	**0**	12	37	51	144	**100**	17	33	50	129	**0**	21	37	42	136	**33**	17	35	48
				2	139	**0**	12	39	49	175	**100**	28	31	41	123	**0**	7	49	44	146	**33**	16	39	45
	2	1	1	1	102	**100**	18	20	62	56	**0**	27	22	51	61	**0**	36	24	40	73	**33**	27	22	51
				2	70	**0**	29	21	50	25	**100**	16	16	68	70	**0**	32	20	48	55	**33**	26	19	55
			2	1	133	**100**	31	30	39	147	**100**	23	29	48	111	**0**	20	25	55	130	**67**	25	28	47
				2	140	**0**	29	31	40	142	**100**	6	32	62	125	**100**	19	28	53	136	**67**	18	30	52
		2	1	1	193	**100**	36	0	64	119	**0**	48	0	52	87	**100**	40	0	60	133	**67**	41	0	69
				2	99	**0**	38	0	62	131	**100**	45	0	55	93	**100**	37	0	63	108	**67**	40	0	60
			2	1	130	**100**	71	0	29	124	**100**	62	0	38	105	**100**	66	0	34	120	**100**	66	0	34
				2	128	**100**	62	0	38	144	**100**	64	0	36	119	**100**	57	0	43	130	**100**	61	0	39
CVR	1	1	1	1	106	**0**	4	25	71	89	**100**	5	36	59	127	**0**	6	39	55	107	**33**	5	33	62
				2	101	**0**	2	27	7	107	**0**	1	33	66	133	**0**	3	40	57	114	**0**	2	33	65
			2	1	99	**0**	5	27	68	139	**0**	7	34	59	131	**100**	8	45	47	123	**33**	7	35	58
				2	115	**0**	3	33	64	128	**0**	5	31	64	144	**0**	3	29	68	129	**0**	4	31	65
		2	1	1	63	**100**	21	19	60	98	**100**	25	27	48	92	**0**	24	33	43	85	**67**	23	27	50
				2	81	**0**	20	22	56	77	**0**	14	30	56	109	**100**	22	35	43	89	**33**	19	29	52
			2	1	89	**100**	20	34	46	100	**0**	24	39	37	122	**100**	29	30	41	104	**67**	24	34	42
				2	121	**0**	19	29	52	98	**100**	27	27	46	117	**0**	25	41	34	112	**33**	24	32	44
	2	1	1	1	131	**100**	24	13	63	127	**100**	31	20	49	119	**100**	22	21	57	126	**100**	26	18	56
				2	144	**100**	24	17	59	130	**100**	27	17	56	149	**0**	24	20	56	141	**67**	25	18	57
			2	1	133	**0**	28	16	56	137	**100**	33	12	55	128	**100**	29	20	51	133	**67**	30	16	54
				2	140	**0**	27	17	56	140	**0**	29	23	52	132	**100**	22	19	59	137	**33**	26	20	54
		2	1	1	94	**100**	32	0	68	146	**100**	36	0	64	131	**100**	31	0	69	124	**100**	33	0	67
				2	159	**100**	29	0	71	99	**100**	35	0	65	144	**100**	27	0	73	134	**100**	30	0	70
			2	1	142	**100**	75	0	25	131	**100**	60	0	40	155	**100**	65	0	35	143	**100**	67	0	33
				2	133	**100**	63	0	37	144	**100**	61	0	39	138	**100**	59	0	41	138	**100**	61	0	39

Terminal attacks of intention IS (Identification of the host Services):

ST, SS, SFI, SX, SN, SU, HS, SFB, DHS, PS,

where ST – “TCP connect scan”, SS – “TCP SYN scan”, SFI – “TCP FIN scan”, SX – “TCP Xmas Tree scan”, SN – “TCP Null scan”, SU – “UDP scan”, HS – “Half scan”, SFB – “Scanning “FTP Bounce””, DHS – “Dumb host scan”, PS – ““Proxy”-scanning”.

Terminal attacks of intention IO (Identification of the host Operating system):

TZ, TS, FF, RF, RS, II, IL, MD, IW, MA, IV, IF, IP, ISP, IDOS,

where TZ – “Connection on telnet and examination of the message header about operating system”, TS – “Connection on telnet and execution of the SYST command (for Unix/Linux)”, FF – “Connection on FTP and examination of bin-files in the directory /bin/ls (for Unix/Linux)”, RF – “FIN Probe - Exploration by the FIN package”, RS – “Bogus flag Probe - Exploration by the package SYN with a false (unused) flag (BOGUS-flag)”, II – “ISN sampling - Capture of initial sequential number ISN at response to a TCP SYN connection request”, IL – “Definition of the law of the ISN change”, MD – “Monitoring of the fragmentation prohibition bit DF”, IW – “Watching of an initial size of the TCP window”, MA – “Watching of value of sequential number used for a field ACK”, IV – “ICMP error message quenching”, IF – “ICMP message quoting”, IP – “Examination of the answer for sending of the TCP packet with certain values of a field “Options””, ISP – “Examination of possibility of "struggle with a flooding" by SYN- packets”, IDOS – “Examination of response for DoS attacks Ping of Death, WinNuke, Teardrop, Land for detection of a Windows OS type (95/98/Me/NT/2000)”.

Terminal attacks of intention CI (Collecting of additional Information):

IST, AM, NS,

where IST – “Inquiry of system time”, AM – “Definition of the network adapter mask”, NS – “Collection of the additional information from DNS-server”.

Terminal attacks of intention RE (Resource Enumeration):

EDNV, EDC, CNS, ERD, SRE, NV, RMT, SRVC, SRVI, DUMP, LEG, NAT, NETD, NETV,

where EDNV – “Enumerating NT/2000 Domains with *net view*”, EDC – “Enumerating NT/2000 Domain Controllers with *nltest*”, CNS – “Connection “null sessions””, ERD – “Enumerating NT/2000 related domains with *nltest*”, SRE – “*Showmount* Resource Enumeration”, NV – “Enumerating NetBIOS Shares with *net view*”, RMT – “Enumerating NetBIOS Shares with *Rmtshare*”, SRVC – “Enumerating NetBIOS Shares with *Srvcheck*”, SRVI – “Enumerating NetBIOS Shares with *Srvinfo –s*”, DUMP – “Enumerating NetBIOS Shares with *DumpSec* (*DumpACL*)”, LEG – “Enumerating NetBIOS Shares with *Legion*”, NAT – “Enumerating NetBIOS Shares with *NetBIOS Auditing Tool* (*NAT*)”, NETD – “Enumerating NetBIOS Shares with *Netdom*”, NETV – “Enumerating NetBIOS Shares with *Netviewx*”.

Terminal attacks of intention UE (Users and groups Enumeration):

DNNT, SNMPE, CNS, FUE, UTFTP, EUE, PIUD, ISU, IAS,

where DNNT – “Dumping the NetBIOS Name Table with *nbtstat* and *nbtscan*”, SNMPE – “SNMP Enumeration with *snmputil* or *IP Network Browser*”, CNS – “Connection “null sessions””, FUE – “Finger Users Enumeration”, UTFTP – “Use of Trivial File Transfer Protocol for Unix enumerating by stealing */etc/passwd* and (or) */etc/hosts.equiv* and (or) *~/.rhosts*”, EUE – “Enumerating Users with *enum*”, PIUD – “Providing Information about Users with *DumpSec* (*DumpACL*)”, ISU – “Identifying SID with user2sid”, IAS – “Identifying Account with sid2user using user’s RID”.

Terminal attacks of intention ABE (Applications and Banners Enumeration):

TCBG, UNU, FP, UREG, UDUM,

where TCBG – “Telnet Connection Banner Grabbing”, UNU – “Use of *netcat* utility for application enumeration”, FP – “FTP-server prompt”, UREG – “Use of *regdmp*”, UDUM – “Use of DumpSec (DumpACL) for output of all services and drivers executed on a host”.

Terminal attacks of intention GAR (Gaining Access to Resources):

CPF, AAF, BFPG, RAH, FCA, PG, AR, UDG, RAM, RA, DIMC, EFE, BO, MMC, UPWS, TH, MP, ABTH, ATH, SF, LA, PF, SA, PD, UF, IFS, APF, WDPF, MUID, MRF, CC,
where CPF – “Cracking of PWL File and access to a host”, AAF – “Anonymity Access to Ftp-server”, BFPG – “Brute Force Password Guessing and access to a host”, RAH – “Replaying the Authentication Hash and access to a host”, FCA – “Free Common Access”, PG – “Password Guessing (for example, with ***BF tool*** of ***Legion***) and access realization”, AR – “Access Realization with permission of recording”, UDG – “User Data Guessing”, RAM – “Register Access and Modification”, RA – “Access to resources”, DIMC – “Direct Implantation of Malicious Code, providing access to resources of a host, in folder %systemroot%\StartMenu\Programs\Startup and start them on execution at the subsequent reboot of the system”, EFE – “External File Execution”, BO – “Buffer Overflow with the subsequent implantation of hostile executed programs”, MMC – “Usage of Malicious Mobile Code”, UPWS – “Usage of initial versions of ***Personal Web Server*** (Microsoft) for gaining files contents and access to a host”, TH – “Trojan horse implantation”, MP – “Mailing password and access to a host”, ABTH – “Access on Behalf of Trusted Host to a host with SunOS v.1.4.x (using the ISN change law)”, ATH – “Access to target host with rlogin”, SF – “SYN flood (storm of inquiries on installation of TCP-connections)”, LA – “Land attack”, PF – “Ping flooding (storm of echoes - inquiries on the ICMP protocol)”, SA – “Smurf attack”, PD – “Ping of Death”, UF – “UDP flooding”, IFS – “Storm of inquiries to FTP-server”, APF – “Access to Password File ***.passwd***”, WDPF – “Writing of Data with user ID to Password File”, MUID – “Modification of user ID”, MRF – “Modification of Rhost File (writing of IP-address of an attacking host)”, CC – “Connection Closing”.

The full set of attacks generated at realization of intention GAR (90 attacks) is as follows:

STIH, SSIH, DC, ST, SS, SFI, SX, SN, SU, HS, SFB, DHS, PS, TZ, TS, FF, RF, RS, II, IL, MD, IW, MA, IV, IF, IP, ISP, IDOS, IST, AM, NS, EDNV, EDC, CNS, ERD, SRE, NV, RMT, SRVC, SRVI, DUMP, LEG, NAT, NETD, NETV, DNNT, SNMPE, CNS, FUE, UTFTP, EUE, PIUD, ISU, IAS, TCBG, UNU, FP, UREG, UDUM, CPF, AAF, BFPG, RAH, FCA, PG, AR, UDG, RAM, RA, DIMC, EFE, BO, MMC, UPWS, TH, MP, ABTH, ATH, SF, LA, PF, SA, PD, UF, IFS, APF, WDPF, MUID, MRF, CC.

The list of attacks removed from the full set of attacks (27 attacks (30 %)), intended for formation of the list of the attacks forbidden by network firewall, is as follows:

SX, TS, FF, IDOS, IST, DNNT, SNMPE, AR, UDG, UREG, UDUM, FUE, UTFTP, EUE, PIUD, ISU, IAS, RAM, RA, DIMC, MMC, UPWS, LA, PF, SA, MRF, CC.

The list of attacks removed from the full set of attacks (36 attacks (40 %)), intended for formation of the list of the attacks forbidden by personal firewall, is as follows:

SSIH, DC, ST, RS, II, IL, MD, IW, MA, CNS, ERD, SRE, NV, RMT, NETV, CNS, TCBG, UNU, FP, MP, ABTH, ATH, SF, PD, TH, UF, IFS, APF, SRVI, DUMP, LEG, NAT, NETD, CPF, AAF, WDPF.

At carrying out the attacks realizing intention GAR, it was supposed, that depending on protection degree a *network firewall* can block the following terminal level attacks:

1) For "Strong" protection degree from full set of the attacks generated at intention GAR realization, the following 63 attacks (70 %) are chosen:

STIH, SSIH, DC, ST, SS, SFI, SN, SU, HS, SFB, DHS, PS, TZ, RF, RS, II, IL, MD, IW, MA, IV, IF, IP, ISP, AM, NS, EDNV, EDC, CNS, ERD, SRE, NV, RMT, SRVC, SRVI, DUMP, LEG, NAT, NETD, NETV, CNS, TCBG, UNU, FP, CPF, AAF, BFPG, RAH, FCA, PG, EFE, BO, TH, MP, ABTH, ATH, SF, PD, UF, IFS, APF, WDPF, MUID.

2) For “None”: - .

The *protection degrees of personal firewall* are as follows:

1) For “Strong” protection degree from full set of the attacks generated at intention GAR realization, the following 54 attacks (60 %) are chosen:

STIH, SS, SFI, SN, SU, HS, SFB, DHS, PS, TZ, RF, IV, IF, IP, ISP, AM, NS, EDNV, EDC, SRVC, BFPG, RAH, FCA, PG, EFE, BO, MUID, SX, TS, FF, IDOS, IST, DNNT, SNMPE, AR, UDG, UREG, UDUM, FUE, UTFTP, EUE, PIUD, ISU, IAS, RAM, RA, DIMC, MMC, UPWS, LA, PF, SA, MRF, CC.

2) For “None”: - .

<u>2. Protection parameters of attacked host.</u>

The host `spiiran-erv,` having IP-address 192.168.130.135, has been chosen as an attacked host at intention GAR realization.

Experiments were fulfilled for two degrees of host protection parameters:

1 – “Strong”;

2 – “Weak”.

Main parameters of the host `spiiran-erv` at "Strong" protection parameters of host are submitted in Tab.2.6.5.

Main parameters of the host `spiiran-erv` at "Weak" protection parameters of host are submitted in Tab.2.6.6.

Base protection parameters, which values differ in a configuration "Strong" and "Weak", are selected in tables with grey color.

Tab.2.6.5. Main parameters of the host `spiiran-erv` at “Strong” protection parameters of host

Parameter name	**Parameter value**		
IP-address	192.168.130.135		
Name	spiiran-erv		
Active ports	21, 23, 80, 137, 136, 8080		
Remote Registry	Off		
Null Sessions	Off		
Password Protected Login	Yes		
Name	spiiran-erv		
Password	RtYrw_!@		
SID	4-1-5-25-8378987-1494822062-1827838900-524		
Sharing Files and Printers	No		
Host is Domain Name Server	Yes		
Domain Name	spiiran-erv.lan3.net		
Name	lan3.net		
Administrator	Admin		
Mail Alias	mail.lan3.net		
Hosts	Name	IP	Post
	Firewall	210.122.25.1	
	Vladimir	192.168.130.138	Vladimir.lan3.net
	Oleg	192.168.130.139	Oleg.lan3.net
	Victor	192.168.130.140	Victor.lan3.net
	Igor	192.168.130.141	Igor.lan3.net
OS platform	Windows		
OS name	2000		
OS version	SP3		
Running Applications	MS IIS, FTP-server, Mail-server, DNS-server, WINS-server		
Shared Resources	No		
Trusted Hosts	No		

Tab.2.6.6. Main parameters of the host `spiiran-erv` at “Weak” protection parameters of host

Parameter name	**Parameter value**		
IP-address	192.168.130.135		
Name	spiiran-erv		
Active ports	21, 23, 80, 137, 136, 8080		
Remote Registry	On		
Null Sessions	On		
Password Protected Login	Yes		
Name	spiiran-erv		
Password	RtYrw_!@		
SID	4-1-5-25-8378987-1494822062-1827838900-524		
Sharing Files and Printers	Yes		
Host is Domain Name Server	Yes		
Domain Name	spiiran-erv.lan3.net		
Name	lan3.net		
Administrator	Admin		
Mail Alias	mail.lan3.net		
Hosts	Name	IP	Post
	Firewall	210.122.25.1	
	Vladimir	192.168.130.138	Vladimir.lan3.net
	Oleg	192.168.130.139	Oleg.lan3.net
	Victor	192.168.130.140	Victor.lan3.net
	Igor	192.168.130.141	Igor.lan3.net
OS platform	Windows		
OS name	2000		
OS version	SP3		
Running Applications	MS IIS, FTP-server, Mail-server, DNS-server, MS Remote Registry Service, WINS-server, MS SQL Server 2000, PWS, SNMP-agent		
Shared Resources	Name	Path	
	C	\\ spiiran-erv \C	
	D	\\ spiiran-erv \D	
Trusted Hosts	Name	IP	
	Vladimir	192.168.130.138	
	Oleg	192.168.130.139	
	Victor	192.168.130.140	
	Igor	192.168.130.141	

3. Parameters defining a hacker’s knowledge about a network.

Experiments were fulfilled for two degrees of a hacker’s knowledge about a network:

1 – “Good” (a hacker’s knowledge about the target host `spiiran-erv` is depicted in Tab.2.6.7);

2 – “Nothing” (a hacker knows nothing about network, except for the IP-address of an attacked host).

Tab.2.6.7. Parameters of the host `spiiran-erv`,
about which a hacker knows (degree of knowledge is “Good”)

Parameter name	**Parameter value**		
IP-address	192.168.130.135		
Name	spiiran-erv		
Active ports	21, 23, 80, 137, 136, 8080		
Password Protected Login	Yes		
Name	spiiran-erv		
Host is Domain Name Server	Yes		
Domain Name	spiiran-erv.lan3.net		
Name	lan3.net		
Administrator	Admin		
Mail Alias	mail.lan3.net		
Hosts	Name	IP	Post
	Firewall	210.122.25.1	
	Vladimir	192.168.130.138	Vladimir.lan3.net

Parameter name	Parameter value		
	Oleg	192.168.130.139	Oleg.lan3.net
	Victor	192.168.130.140	Victor.lan3.net
	Igor	192.168.130.141	Igor.lan3.net
OS platform	Windows		
OS name	2000		
OS version	SP3		
Running Applications	MS IIS, FTP-server, Mail-server, DNS-server, WINS-server		

Examples of the screens, displaying various stages of attack scenario generation for intention GAR, are submitted in Fig.2.6.11 – Fig.2.6.13. The values of input parameters used for this attack scenario are as follows:

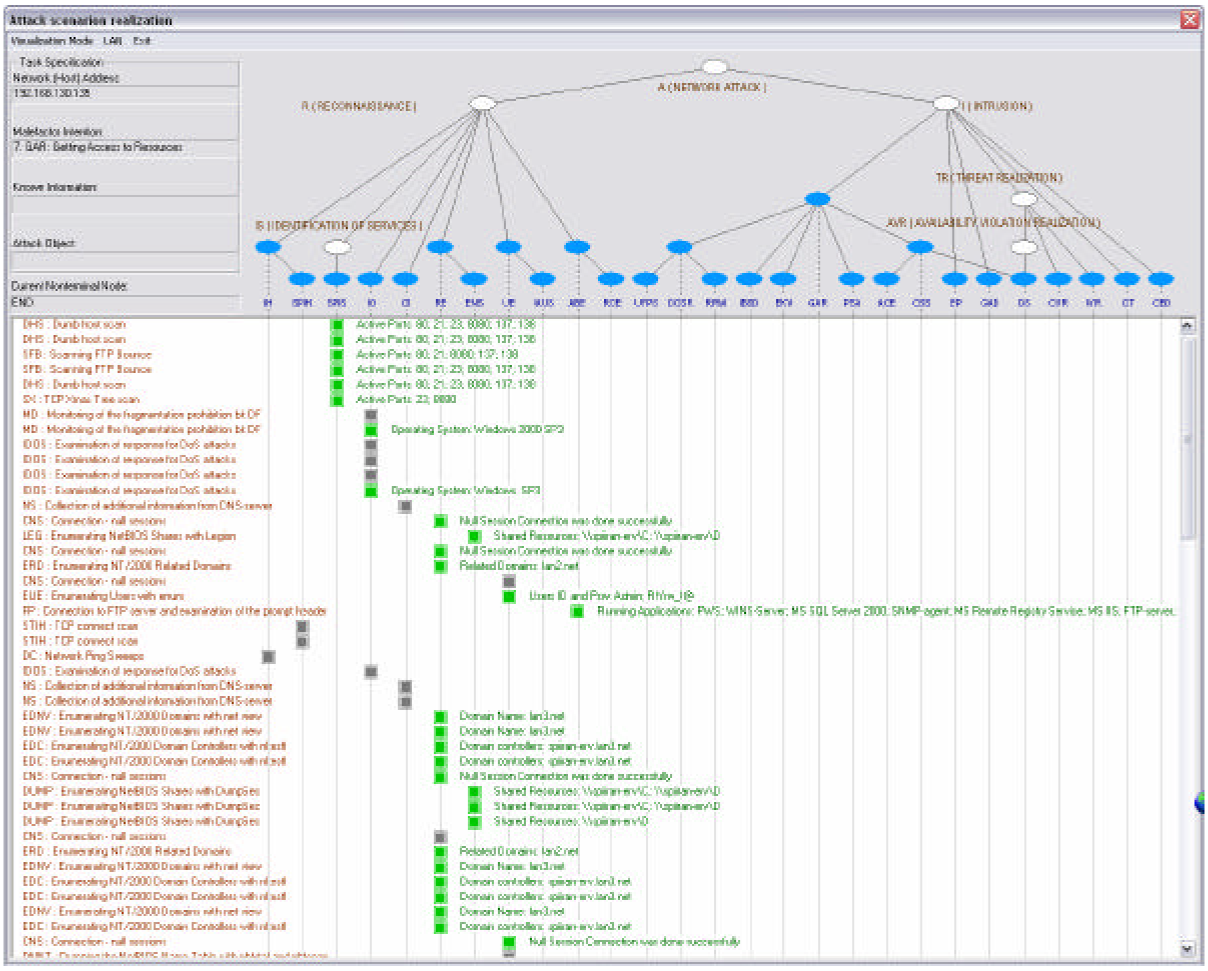
(1) protection degree of network firewall is "None" (2);

(2) protection degree of personal firewall is "None" (2);

(3) protection degree of host parameters is "Weak" (2);

(4) degree of a hacker's knowledge about a network is "Nothing" (2).

As we can see from attack scenario traces depicted in the figures, the majority of attack actions is realized successfully owing to weak protection of an attacked network and a host (successful actions are designated by green color).



Task Specification
Network (Host) Address: 192.168.130.138
Malefactor Intention: 7. GAR: Getting Access to Resources
Known Information:
Attack Object:
Current Nonterminal Node: END

```
DHS : Dumb host scan                                       Active Ports: 80; 21; 23; 8080; 137; 138
DHS : Dumb host scan                                       Active Ports: 80; 21; 23; 8080; 137; 138
SFB : Scanning FTP Bounce                                  Active Ports: 80; 21; 8080; 137; 138
SFB : Scanning FTP Bounce                                  Active Ports: 80; 21; 23; 8080; 137; 138
DHS : Dumb host scan                                       Active Ports: 80; 21; 23; 8080; 137; 138
SX : TCP Xmas Tree scan                                    Active Ports: 23; 8080
MD : Monitoring of the fragmentation prohibition bit DF
MD : Monitoring of the fragmentation prohibition bit DF    Operating System: Windows 2000 SP3
IDOS : Examination of response for DoS attacks
IDOS : Examination of response for DoS attacks
IDOS : Examination of response for DoS attacks
IDOS : Examination of response for DoS attacks             Operating System: Windows SP3
NS : Collection of additional information from DNS-server
CNS : Connection - null sessions                           Null Session Connection was done successfully
LEG : Enumerating NetBIOS Shares with Legion                 Shared Resources: \\spiirin-srv\C; \\spiirin-srv\D
CNS : Connection - null sessions                           Null Session Connection was done successfully
ERD : Enumerating NT/2000 Related Domains                  Related Domains: lan2.net
CNS : Connection - null sessions
EUE : Enumerating Users with enum                            Users ID and Priv. Admin: Rf/rw_!@
FP : Connection to FTP server and examination of the prompt heads   Running Applications: PWS; WINS-Server; MS SQL Server 2000; SNMP-agent; MS Remote Registry Service; MS IIS; FTP-server;
STIH : TCP connect scan
STIH : TCP connect scan
DC : Network Ping Sweeps
IDOS : Examination of response for DoS attacks
NS : Collection of additional information from DNS-server
NS : Collection of additional information from DNS-server
EDNV : Enumerating NT/2000 Domains with net view           Domain Name: lan3.net
EDNV : Enumerating NT/2000 Domains with net view           Domain Name: lan3.net
EDC : Enumerating NT/2000 Domain Controllers with nltest   Domain controllers: spiirin-srv.lan3.net
EDC : Enumerating NT/2000 Domain Controllers with nltest   Domain controllers: spiirin-srv.lan3.net
CNS : Connection - null sessions                           Null Session Connection was done successfully
DUMP : Enumerating NetBIOS Shares with DumpSec               Shared Resources: \\spiirin-srv\C; \\spiirin-srv\D
DUMP : Enumerating NetBIOS Shares with DumpSec               Shared Resources: \\spiirin-srv\C; \\spiirin-srv\D
DUMP : Enumerating NetBIOS Shares with DumpSec               Shared Resources: \\spiirin-srv\D
CNS : Connection - null sessions
ERD : Enumerating NT/2000 Related Domains                  Related Domains: lan2.net
EDNV : Enumerating NT/2000 Domains with net view           Domain Name: lan3.net
EDC : Enumerating NT/2000 Domain Controllers with nltest   Domain controllers: spiirin-srv.lan3.net
EDC : Enumerating NT/2000 Domain Controllers with nltest   Domain controllers: spiirin-srv.lan3.net
EDNV : Enumerating NT/2000 Domains with net view           Domain Name: lan3.net
EDC : Enumerating NT/2000 Domain Controllers with nltest   Domain controllers: spiirin-srv.lan3.net
CNS : Connection - null sessions                             Null Session Connection was done successfully
```

Fig.2.6.11. Example of the screen displaying the attack scenario generation processes of the intention GAR (an initial stage of attack scenario)